Bachelor Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Robotic Trajectory Optimization

Petr Cezner

Supervisor: Ing. Martin Ron
Field of study: Cybernetics and Robotics
Subfield: Systems and Control
May 2017

Acknowledgements

I would like to thank my supervisor Ing. Martin Ron for his leadership, help and wise advice during the writing of this thesis.

I would also like to thank my family and all of my friends for their support during the writing of this work and during my studies.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in preparation of university theses

In Praque, 24. May 2017

. . . .

Abstract

With the onset of the Fourth Industrial Revolution, an effort is being made to optimize existing production processes to maximize production. One area where attempts are made to optimize production are industrial complexes with robot manipulators. In this work, an algorithm is implemented which, from the knowledge of the internal structure of robots (their dynamics) is able to optimize trajectory of their end-effectors in order to achieve lower energy consumption. The algorithm using several points in space (ie. via-points) of the original collision-free trajectory forms a new energy-efficient trajectory. The added value of the thesis is the automatic creation of a program for the robot written in language KRL for KUKA robots. This program has been successfully tested on a real robot and resulting reduction of power was to 14.48 percent of the original one.

Keywords: Optimization, Energy demand, Manipulator, Trajectory, IPOPT solver, Matlab, KUKA, KRL code

Supervisor: Ing. Martin Ron

Abstrakt

S nástupem čtvrté průmyslové revoluce se čím dál tím víc projevuje snaha optimalizovat stávající výrobní procesy pro dosažení maximální produkce. Jednou z oblastí, kde je snaha optimalizovat výrobu jsou průmyslové komplexy s robotickými manipulátory. V této práci je implementován algoritmus, který ze znalosti vnitřní struktury robotů (jejich dynamiky) je schopen optimalizovat jejich trajektorii s cílem dosažení nižší energetické spotřeby. Algoritmus pomocí několika bodů v prostoru (tzv. via-points) původní bezkolizní trajektorie koncového efektoru vytvoří novou energeticky účinnější trajektorii. Přidanou hodnotou předložené práce je automatická tvorba programu pro pohyb robotu v jazyku KRL pro KUKA roboty. Tento program byl úspěšně testován na reálném robotu a výsledné trajektorie dosahovaly snížení energetické náročnosti až 14.48 procent původní spotřeby.

Klíčová slova: Optimalizace, Energetická náročnost, Manipulátor, Trajektorie, IPOPT solver, Matlab, KUKA, KRL kód

Překlad názvu: Optimalizace robotických trajektorií

Contents

1 Introduction 1

2 State of the Art 3
2.1 Used method 3

3 Theoretical basis 5
3.1 Optimization 5
  3.1.1 Optimization Problem 5
  3.1.2 Solving Optimization Problem 5
  3.1.3 Nonlinear Optimization 6
  3.1.4 IPOPT Solver 7
3.2 Mathematical model of a Robotic Manipulator 7
3.3 Kinematics Model 7
  3.3.1 Daneavit-Hartenberg (DH) Convention 8
  3.3.2 Direct kinematic 9
3.4 Dynamic Model 10
  3.4.1 Euler-Lagrange formulation vs. Newton-Euler formulation 10
  3.4.2 Newton-Euler formulation . . . 11
3.5 Studied System 14
  3.5.1 Robot Parameters 14

4 Solving the problem 19
4.1 Problem setup 19
  4.1.1 Optimization variables 20
  4.1.2 Geometrical Optimalization . 21
  4.1.3 Kinematic Optimization 23
4.2 Implemation of the Criterion function 25
  4.2.1 Calculation of the torques . . . 25
  4.2.2 Calculation of the currents . . 26
  4.2.3 Evalution of the Criterion function 27

5 Experiments 31
5.1 Experiment in Simulated Environment 31
  5.1.1 Original Trajectory 31
  5.1.2 Optimal Trajectory 32
  5.1.3 Results from simulation 33
5.2 Experiment with real manipulator 42
  5.2.1 Trajectory 1 43
  5.2.2 Trajectory 2 47

6 Conclusion 51
6.1 Future improvements 52

A Bibliography 53

B Figures from Experiments 55
B.1 Trajectory 1 - Simulation 55

C Content of the CD 67

D Project Specification - English Version 69

E Project Specification 71

Figures

3.1 Denavit–Hartenberg kinematic parameters - Siciliano et al. [14] . . . 8
3.2 Link i in Newton-Euler formulation - Siciliano et al. [14] . . 11
3.3 KUKA KR5 ARC 16
3.4 Operation space of KUKA KR5 - part 1 KUKA Roboter Group GmbH [10] 16
3.5 Operation space of KUKA KR5 - part 2 KUKA Roboter Group GmbH [10] 17

4.1 Trajectory of the manipulator end-effector. 20
4.2 Evaluation of s in polynomials (4.18), (4.19) and (4.20) 23
4.3 Bond graph of motor i 26

5.1 Computed position of original Trajectory 1. The black arrow shows direction of the movement. 34
5.2 Computed position of optimazed Trajectory 1. The black arrow shows direction of the movement. 35
5.3 Trajectory 1 - side view from Process Simulate 35
5.4 Trajectory 1 - front view from Process Simulate 36
5.5 Close look up at speed and acceleration of first joint - original trajectory 37
5.6 Close look up at speed and acceleration of first joint - Optimal one 37
5.7 Computed power in each discretization step - original trajectory 38
5.8 Computed power in each discretization step 38
5.9 Computed position of original Trajectory 2. The black arrow shows direction of the movement. 39
5.10 Computed position of optimazed Trajectory 2. The black arrow shows direction of the movement. 40
5.11 Close look up at speed and acceleration of first joint - original trajectory 40
5.12 Close look up at speed and acceleration of first joint - Optimal one 41
5.13 Computed power in each discretization step - original trajectory 2 41
5.14 Computed power in each discretization step - optimal trajectory 2 42
5.15 Measured data of trajectory 1 - original one 44
5.16 Measured data of trajectory 1 - optimal one 45
5.17 Power consumption measured on real robot and its interpolation - original trajectory 1 45
5.18 Power consumption measured on real robot and its interpolation - optial trajectory 1 46
5.19 Computed power in each discretization step - optimal trajectory from simulation 46
5.20 Measured power consumption of original second trajectory 47
5.21 Measured power consumption of optimal second trajectory 48
5.22 Closer look at power consumption one cycle of second original trajectory 48
5.23 Closer look at power consumption one cycle of second optimal trajectory 49
5.24 Computed power from application - Trajectory 2 49

B.1 Original trajectory 1 between joint 1 and 2 55
B.2 Original trajectory 1 between joint 1 and 3 56
B.3 Original trajectory 1 between joint 2 and 3 56
B.4 Acceleration of manipulator - original trajectory 1 57

B.5 Calculated power of manipulator - original trajectory 1 57
B.6 Calculated energy of manipulator - original trajectory 1 58
B.7 Speed of manipulator - original trajectory 1 58
B.8 Speed and acceleration of first joint - original trajectory 1 59
B.9 Calculated torque - original trajectory 1 59
B.10 Used power (Blue line) and power caused by movement (orange line) - original trajectory 1 60
B.11 Optimal trajectory between joint 1 and 2 60
B.12 Optimal trajectory between joint 1 and 3 61
B.13 Optimal trajectory between joint 2 and 3 61
B.14 Acceleration of manipulator - optimal trajectory 1 62
B.15 Calculated power of manipulator - optimal trajectory 1 62
B.16 Calculated energy of manipulator - optimal trajectory 1 63
B.17 Velocity of manipulator - optimal trajectory 1 63
B.18 Velocity and acceleration of first joint - optimal trajectory 1 64
B.19 Calculated torque - trajectory 1 64
B.20 Used power (Blue line) and power caused by movement (orange line) - optimal trajectory 1 65

Tables

3.1 Robot parameters - limits of joints 15
3.2 Robot parameters - masses of links 15
3.3 Robot parameters - Inertia vectors for each axis 15
3.4 Robot parameters - Center of the Mass vectors 15

4.1 Total number of variables concerning robot trajectory 21
4.2 Found Daneavit-Hartenberg parameters 26
4.3 Parameters of robot motor 27

5.1 Energy save for simulation - Trajectory 1 33
5.2 Distance between points of original Trajectory 1 (see Figure 5.1) 33
5.3 Energy save for simulation - Trajectory 2 39
5.4 Distance between points of Trajectory 2 (see Figure 5.9) 39
5.5 Values from experiments with real manipulator KUKA KR5 - Trajectory 1 44
5.6 Values from experiments with real manipulator KUKA KR5 - Trajectory 2 47

Chapter 1

Introduction

With the onset of the Industrial Revolution 4.0, there are more efforts to maximize the capacity of manufacturing processes with their cost minimalization. One of the areas, where these efforts could be found are areas which used industrial manipulators in a production. Industrial manipulators are one of the highest electrical energy consumers in the facility. There are several ways how to optimized the energy consumption of the manipulators.

One way is to slow down the speed of manipulator movement. This type of optimization has the disadvantage that the production cell (part of the production line) has to be optimized at once. Due to slow down of movement it is possible that the production cycle time could not be fulfilled.

The other way of optimization is an optimization of robotic trajectories. The implementation of this method, reduces the energy consumption of the robotic manipulators and simultaneously fulfills the production cycle time. The application of this approach needs to assembly the mathematical model of the manipulator. The model brings the understanding of the system dynamics.

The aim of this thesis is to implement the second approach for energy saving. Although the approach is more complex than the first one, it increases the number of possible real applications. The mathematical model is used for the creation of the criteria function. The function is then used as the input into the nonlinear programming (NLP) solver. The basic variables such as position, velocity, and acceleration of the new trajectory are found by the equations presented in the recommended literature of this work.

Chapter 2

State of the Art

Based on the review of the papers published by IEEE (see a survey of literature) it is shown that the full potential of industrial robots isn't fully exploited. In most cases, the industrial manipulators are constructed for the satisfaction of the given task regardless of their power consumption.

The three papers Bukata et al. [3], Björkenstam et al. [1] and Gleeson et al. [5] address this issue for industrial robotic manipulators.

In paper Bukata et al. [3], the method of energy optimization of robotic cells is introduced. This method reduces of the robot velocity without the need for the structure of the manipulator. This approach has highly limitations (for example: not respecting of the robot dynamics, the high complexity of implementation) for applicability in the industry.

In paper Björkenstam et al. [1] and Gleeson et al. [5] the method for optimization of the robotic trajectory is presented. The knowledge of the manipulator structure is required.

2.1 Used method

In the recommended literature Björkenstam et al. [1] and Gleeson et al. [5], there are two algorithms, which can be used for optimization of the robotic trajectory. In general, both of the algorithms are the same. The main difference between them is the number of used variables.

In paper Björkenstam et al. [1], one of the control variables is an actuator (motor) moment. As it is said in Gleeson et al. [5], it is not easy to control this variable on the real robotic manipulator, because the input parameters of the systems are limited by predefined sets of commands. These commands are typically, the position of the point and the velocity of the robot at that point. Probably for that limitations, the original developers of the algorithm for paper Björkenstam et al. [1] came with the new algorithm.

The new algorithm, which is presented in paper Gleeson et al. [5], is more suitable for application in the industry environment. The developers reduce the number of variables used for NLP (non-linear programming) at each point of trajectory from five variables to four.

In paper Björkenstam et al. [1] and GleesonGleeson et al. [5] are used the same discrete optimal control problem as of the continuous optimal problem, and as stated in Gleeson et al. [5], paper Gleeson et al. [5] is the continuation of the work presented in Björkenstam et al. [1]. For that reason, in the following chapter, the algorithm presented in Gleeson et al. [5] will be explored and implemented.

Chapter 3

Theoretical basis

In this chapter, the fundamental theory for optimizations of robotic trajectory is introduced. The first part of this chapter 3.1 deals with the optimization. This section explains how the proper optimization problem is setup and how the different algorithms are used for solving. The second part of the chapter 3.2 is about the Mathematical model of the robotic manipulator. The key terms like the kinematic model 3.3 and the dynamic model 3.4 are introduced.

3.1 Optimization

Optimization is the process of best parameters selection from common set of available parameters. The goal of the optimization task is to find a set of values of the otimized variables which maximize or minimize criterial function.

3.1.1 Optimization Problem

According to Boyd and Vandenberghe [2], a mathematical optimization problem has the form:

$$\min_x f_0(\mathbf{x}) \tag{3.1}$$

$$\text{subject to } f_i(\mathbf{x}) \leq b_i \quad i = 1, \cdots, m, \tag{3.2}$$

where the $\mathbf{x} = (x_1, \cdots, x_n)$ is the optimization variable of the problem, the function f_0: $\mathbf{R}^n \to \mathbf{R}$ is the objective function, the functions f_i: $\mathbf{R}^n \to \mathbf{R}$, $i = 1, \cdots, m$, are the constraint functions. There are two type of constraints functions: equality and inequality. The functions (3.2) are inequality constraints functions. These functions have limits, or bounds. In (3.2) $b_1, \cdots, b_m$ are limits for inequality functions. A vector $\mathbf{x}^*$ is called the optimal, or a solution of the problem (3.1)-(3.2), if it has the smallest objective value among all vectors that satisfy the constraints: for any z with $f_1(z) \leq b_1, \cdots, f_m(z) \leq b_m$, we have $f_0(z) \geq f_0(\mathbf{x}^*)$.

3.1.2 Solving Optimization Problem

To solve the optimization problem, algorithm which calculates a solution of the problem (to some given accuracy) is used. If constraints like in (3.1)-

(3.2) are set, the algorithm (solver) has to satisfy it. In these days, there are many of solvers, and each of them is suitable for a different kind of optimization problem. The effectiveness of these algorithms, as stated in Boyd and Vandenberghe [2], or the ability to solve the optimization function problem, depends on many factors. For example the particular form of the objective and constraints functions, number of input variables and constraints, there are some unique structures, such as sparsity (as is said in Boyd and Vandenberghe [2]: „Problem is *sparse* if each constraint function depends on only a small number of variables“).

3.1.3 Nonlinear Optimization

If the object or constraints functions are not linear, but not known to be convex, nonlinear optimization (nonlinear programming) is used. Unfortunately, the general nonlinear problem is difficult to be effectively solved. In the real world, we can find many systems, which are inherently nonlinear, e.g. in the case of electrical circuit, the nonlinear element is a diode. For the diode, classical Ohm law doesn't be apply, and if the diode is in the complex electrical circuit, it is not possible to solve this problem with traditional method (superposition, loop currents,...). So it is crucial to have the algorithms, which are capable of handling it. According to Chinneck [4], there are twelve main reasons, why nonlinear models are inherently much more difficult to optimized. First, seven of them are presented here:

1. **Differing between local and global optimum is difficult.** The main limitation of numerical methods for solving nonlinear programs is that they usually have information only about the current point. Typically they have information about current point x (and stored information about past points that have been visited), about the value of the function at x, the value of the constraint functions at x and so on. This limited information is just enough for recognizing when the function is at local maximum or minimum. This also means that there is not enough imformation to find global maxima.

2. **Optima are not restricted to extreme points.** In linear programming, there are limited places to look for the optimum. It only checks to the extreme points or corner points of the feasible region. For nonlinear programming, an optimum (local or global) could be anywhere. The optimum could be at an extreme point, along with the edge of the feasible area, or in the interior of the feasible region.

3. **There could be many disconnected possible areas.** Nonlinear constraints, which are used, may twist and curve the areas so they can create multiple possible regions. So if the algorithm finds a suitable area, there could be some other disconnected (discontiguous) feasible regions that the algorithm has not found and explore.

4. **Different starting points may lead to different final solutions.** Many algorithms choose a direction for search and then find the best value of the objective function in that direction.

5. **Finding a feasible starting point may be difficult.** Unlike in linear programming in the nonlinear programming it is not guaranteed finding a point that satisfies all of the constraints and hence is feasible. It may happen that no feasible points exist anywhere.

6. **Satisfy equality constraints is difficult.** In nonlinear programming, the solution is difficult to find, which meets curving and twisting equations of the constraint functions. Equality constraints may be violated if the algorithm moves to the another point, which has a better solution for objective function.

7. **Determination of the outcome is not designated.** There is no guarantee that the solution found by solver is local optimum or global optimum. The calculation of the objective function value could take a long time and could end like an infeasible solution. In these cases, it is not strictly necessary that the object function is infeasible if no feasible solution is found.

3.1.4 IPOPT Solver

In this thesis, IPOPT solver is used. As in Gleeson et al. [5] the IPOPT solver was used. Abbreviation IPOPT stands for Interior Point Optimizer. It is open source software package for large-scale nonlinear optimization as is said in Kawajir [8]. IPOPT implements an interior line search filter method that aims to find a local solution of (3.1)-(3.2).

3.2 Mathematical model of a Robotic Manipulator

For better adjusting the robot controllers, kinematic and dynamic mathematical model of the robotic manipulator is needed. In following section the mathematical model is introduced.

3.3 Kinematics Model

As is said in Siciliano et al. [14] and Grepl [6], from mechanical viewpoint the robotic manipulator is viewed as a kinematic chain of rigid bodies (links). These links are connected to each other by joints. There are two types of joints: revolute and prismatic. The manipulator is specified as chain having one end constrained to the base of the robot. Another end is called end-effector. The motion of each link of the manipulator is determined by the motion of previous one. It is necessary to know the position and orientation of the end-effector for manipulating the object in space. In this section, the fundamental information about direct kinematic is introduced.

3.3.1 Daneavit-Hartenberg (DH) Convention

In robotics, the determination of the position and the orientation of base of reference is ambiguous. There are several options. The shape of transformation matrices is depends on the chosen option. The general systematic method for describing a respective position links of the robot is needed. The best option is the solution with the smallest number of parameters. In robotics, the Denavit Hartenberg (or DH) convention for the description of the open kinematic chain is used. DH convention uses four variables. In the Figure 3.1, we can see two rigid bodies i and $i-1$ connected to each other with rotation linkage. Let's assume that the joints are twisted.

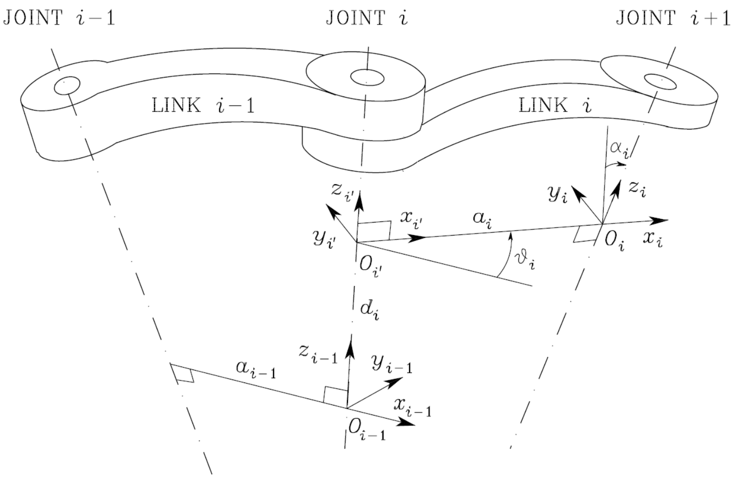

Figure 3.1: Denavit–Hartenberg kinematic parameters - Siciliano et al. [14]

Denavit Hartenberg convention is adopted to define the state of the link of frame i as follows Siciliano et al. [14]:

1. Choose axis z_i along the axis of Joint $i + 1$
2. Locate the origin O_i at the intersection of axis of axis z_i with the common normal to axes z_{i-1} and z_i. Also, locate $O_{i'}$ at the intersection of the common normal with axis z_{i-1}.
3. Choose axis x_i along the common normal to axes z_{i-1} and z_i with direction from Joint i to Joint $i + 1$.
4. Choose axis y_i so as to complete a right-handed frame.

The position and orientation of the Frame i on the Frame $i+1$ are completely specified by specifying the four parameters:

1. a_i distatnce between O_i and $O_i^{'}$,
2. d_i coordiante of $O_i^{'}$ along z_{i-1},

3. α_i angle between axes z_{i-1} and z_i about axis x_i to be taken positive when rotation is made counter-clockwise,
4. υ_i angle between axes x_{i-1} and x_i about axis z_{i-1} to be taken positive when rotation is made counter-clockwise.

Two of the four parameters (a_i and α_i) are constant. They are defined by the construction of the manipulator. Remaining parameters υ_i and d_i are the variables depending on the type of the joint between Link i and Link i-1.

1. If the Joint i is **revolute**, then the variable is υ_i.
2. If the Joint i is **prismatic**, then the variable is d_i.

It could be said that, the three out of four parameters are constraints depending on the constructions of the manipulator. The last parameter is depending on the position of the manipulator. As it is said in Siciliano et al. [14], it is possible to use DH parameters to express the transformation between Frame i and Frame $i-1$ by using following transformation matrices.

$$A_{i'}^{i-1} = \begin{pmatrix} \cos(\upsilon_i) & -\sin(\upsilon_i) & 0 & 0 \\ \sin(\upsilon_i) & \cos(\upsilon_i) & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

$$A_i^{i'} = \begin{pmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos(\alpha_i) & -\sin(\alpha_i) & 0 \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

Transformation matrix from Frame i to Frame $i-1$ is the result of multiplication of matricis (3.3) and (3.4).

$$\begin{aligned} A_i^{i-1}(q_i) &= A_{i'}^{i-1} A_i^{i'} \\ &= \begin{pmatrix} \cos(\upsilon_i) & -\sin(\upsilon_i)\cos(\alpha_i) & \sin(\upsilon_i)\sin(\alpha_i) & a_i\cos(\upsilon_i) \\ \sin(\upsilon_i) & \cos(\upsilon_i)\cos(\alpha_i) & -\cos(\upsilon_i)\sin(\alpha_i) & a_i\sin(\upsilon_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{3.5}$$

As it is said in Siciliano et al. [14] and it could be seen in (3.5), the transformation matrix is the function of one input variable q_i. Variable q_i is υ_i for revolute joint or d_i for prismatic joint.

3.3.2 Direct kinematic

The transformation matrix (3.5) described in the previous section can be used for calculation of the direct kinematic problem. Direct kinematic is a process of determination of the position of end-effector given the rotation angle of each joint. The process of direct kinematic always leads to one solution. It leads to one homogenous transformation matrices by the composition of the individual transformation matrices described in (3.5). Input for the function are joint variables q. The outputs are Cartesian coordinates XYZ and corresponding rotation of end-effector. With the use of (3.5), it is possible to calculate transformation from the base of the manipulator to its end-effector.

3.4 Dynamic Model

The kinematic model of robot manipulator considers only the motion without the forces and torques, which act on the robot during its movement, whereas the dynamic model describes the relations between forces, torques and motions Spong et al. [16]. Using the dynamic model, simulation of the real manipulator can be created. In this work, the dynamic model is used for the simulation of movement and energy consumption of the robot. As it is said in Siciliano et al. [14], the dynamic model is created in joint space. There are two methods used for calculation of the dynamic of the robotic manipulator. One is based on Euler-Lagrange formulation, and the other one is based on Newton-Euler formulation. Both of these formulations lead to the same equation of motion (see in Siciliano et al. [14], Spong et al. [16] and GleesonGleeson et al. [5]):

$$\boldsymbol{\tau} = \mathbf{M}(\boldsymbol{q}(t))\ddot{\boldsymbol{q}}(t) + \mathbf{C}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)) + \mathbf{G}(\boldsymbol{q}(t), \dot{\boldsymbol{q}}(t)), \tag{3.6}$$

where $\boldsymbol{\tau}$ is vector of torques, $\mathbf{M}$ is matrix of inertia, $\mathbf{C}$ includes the centrifugal and Coriolis forces, $\mathbf{G}$ is matrix of the external forces i. e. gravity. Variable $\boldsymbol{q}(t)$ is joint space coordinate vector at time t, $\dot{\boldsymbol{q}}(t)$ is joint space vector of speed at time t and $\ddot{\boldsymbol{q}}(t)$ is joint space acceleration vector at time t.

3.4.1 Euler-Lagrange formulation vs. Newton-Euler formulation

It can be seen in section 3.4, the both method leads to the same solution. The question, which formulation should be used is the wildly discussed topic. There is no clear answer, which of these formulations should be applied. There are several factors, which have to be considered during the decision. Euler-Lagrangian formulation look at the multi-body robot as a whole. It also eliminates the inertial reaction forces between the links and is best suited for the analysis of control schemes. On the other hand, the Newton-Euler formulation creates dynamic equations for each link separately. It performs the inverse dynamics in real time by the evaluation of equations in a numeric and recursive way. The Newton-Euler formulation is widely used in robots controllers for its simple implementation.

For the calculation of dynamic equations, the Matlab algorithm called ReDySim (Shah [13]) is used. ReDySim (Recursive Dynamics Simulator) is free-to-use sets of Matlab scripts, which computed the inverse (like Newton-Euler formulation) and forward dynamic recursive equations of the robots and multibody systems. In the next section 3.4.2 the Newton-Euler formulation will be described.

3.4.2 Newton-Euler formulation

This section is based on notation and formulation from Siciliano et al. [14]. In general, the Newton-Euler formulation bases on that, all forces acting on the generic link of the manipulator are balanced. Sets of equations, which are determined by this balance, allows to use recursive type of solution. First, the forward recursion is used to find out the link velocities and accelerations. After that, the backward recursion is used to calculate the force and torque, which acts on the link during its movement. Let us consider the following Link i (Figure 3.2) from the kinematic chain with the motor based on joint $i+1$.

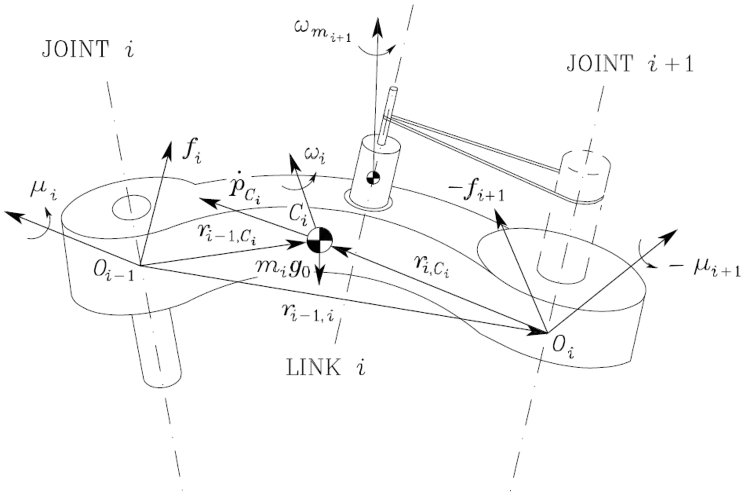

Figure 3.2: Link i in Newton-Euler formulation - Siciliano et al. [14]

The Link has following parameters:

- m_i mass of link
- $\bar{\mathbf{I}}_i$ inertia tensor of link
- I_{m_i} moment of inertia of rotor along its rotational axis
- $\mathbf{r}_{i-1,C_i}$ vector from origin of Frame $(i-1)$ to center of mass C_i
- $\mathbf{r}_{i,C_i}$ vector from origin of Frame (i) to center of mass C_i
- $\mathbf{r}_{i-1,i}$ vector from origin of Frame $(i-1)$ to origin Frame i

The velecoties and accelerations which are considered are:

- $\dot{\mathbf{p}}_{C_i}$ linear velocity of center of mass C_i
- $\dot{\mathbf{p}}_i$ linear velocity of origin of Frame i

- $\boldsymbol{\omega}_i$ angular velocity of link
- $\boldsymbol{\omega}_{m_i}$ angular velocity of rotor
- $\ddot{p}_{C_i}$ linear acceleration of origin of Frame i
- $\dot{\omega}_i$ angular acceleration of link
- $\dot{\omega}_{m_i}$ angular acceleration of rotor
- $\mathbf{g}_0$ gravity acceleration

The forces and moments to be considered are:

- $\mathbf{f}_i$ force exerted by Link $i-1$ on Link i
- $-\mathbf{f}_{i+1}$ force exerted by Link $i+1$ on Link i
- $\boldsymbol{\mu}$ moment exerted by Link $i-1$ on Link i with respect to origin of Frame $i-1$
- $-\boldsymbol{\mu}_{i+1}$ moment exerted by Link $i+1$ on Link i with respect to origin of Frame i

Let us consider that all variables (vectors and matrices) are referenced to the base Frame.

Recursive Algorithm

The derivation of all equations which will be introduced in this subsection can be found in Siciliano et al. [14]. As it is said in Siciliano et al. [14], the recursive algorithm of the Newton-Euler method is not in the closed form. It is because the motion of Link i is computed by the kinematic relationship of velocities and accelerations through the movement.

The joint space positions, velocities, and accelerations have to be known (calculated) before the calculation of the links velocities and accelerations. This calculation is recursive because the velocities and accelerations of the base frame link are calculated first. After that, the Newton-Euler equations for calculating the forces and torques, which affect the links, can be used in the recursive fashion. Starting with the forces and moments of the end-effector and ending with the link of the base frame. As is said in Siciliano et al. [14], in summary, the recursive algorithm can be divided into two parts. Forward which propagates the velocities and accelerations and the backward which spreads the forces and torques along the structure of the kinematic chain.

This work focuses on rotational robotic manipulator, thus only the equations for revolute joints will be considered. Equations for prismatic can be found in Siciliano et al. [14].

$$\boldsymbol{\omega}_i^i = \mathbf{R}_i^{i-1^\top}(\boldsymbol{\omega}_{i-1}^{i-1} + \dot{\vartheta}_i \boldsymbol{z}_0) \tag{3.7}$$

Equation (3.7) compute the angular velocity of Link i. In equation (3.7), $\mathbf{R}_i^{i-1}$ is rotation matrix from previos Link $i-1$ to the Link i. $\boldsymbol{\omega}_{i-1}^{i-1}$ is angular velocity in Link $i-1$, $\dot{\vartheta}_i$ is joint space velocity and $\boldsymbol{z}_0$ is an axis of rotation.

Next equations show the calculation of angular acceleration in Link i:

$$\dot{\boldsymbol{\omega}}_i^i = \mathbf{R}_i^{i-1^\top}(\dot{\boldsymbol{\omega}}_{i-1}^{i-1} + \ddot{\vartheta}_i \boldsymbol{z}_0 + \dot{\vartheta}_i(\boldsymbol{\omega}_{i-1}^{i-1} \times \boldsymbol{z}_0), \tag{3.8}$$

where $\mathbf{R}_i^{i-1}$ is rotation matrix from previos Link $i-1$ to the Link i. $\boldsymbol{\omega}_{i-1}^{i-1}$ is angular velocity in Link $i-1$, $\dot{\boldsymbol{\omega}}_{i-1}^{i-1}$ is angular acceleration in Link $i-1$, $\dot{\vartheta}_i$ is joint space velocity in Link i, $\ddot{\vartheta}_i$ is joint space acceleration and $\boldsymbol{z}_0$ is axis of rotation.

Following equations compute the linear acceleration:

$$\ddot{\boldsymbol{p}}_i^i = \mathbf{R}_i^{i-1^\top}\ddot{\boldsymbol{p}}_{i-1}^{i-1} + \dot{\boldsymbol{\omega}}_i^i \times \boldsymbol{r}_{i-1,i}^i + \boldsymbol{\omega}_i^i \times (\boldsymbol{\omega}_i^i \times \boldsymbol{r}_{i-1,i}^i), \tag{3.9}$$

where $\mathbf{R}_i^{i-1}$ is rotation matrix from previos Link $i-1$ to the Link i. $\ddot{\boldsymbol{p}}_{i-1}^{i-1}$ is linear accelertion of Link $i-1$, $\boldsymbol{\omega}_i^i$ is angular velocity in Link i, $\dot{\boldsymbol{\omega}}_i^i$ is angular acceleration in Link i and $\boldsymbol{r}_{i-1,i}^i$ is vector from origin of Frame $i-1$ to origin of Frame i.

Equation (3.10) show the linear acceleration of the center of the mass (COM) of Link i:

$$\ddot{\boldsymbol{p}}_{Ci}^i = \ddot{\boldsymbol{p}}_i^i + \dot{\boldsymbol{\omega}}_i^i \times \boldsymbol{r}_{i,C_i}^i + \boldsymbol{\omega}_i^i \times (\boldsymbol{\omega}_i^i \times \boldsymbol{r}_{i,C_i}^i), \tag{3.10}$$

where $\ddot{\boldsymbol{p}}_i^i$ is linear acceleration of Link i, $\dot{\boldsymbol{\omega}}_i^i$ is angular acceleration of Link i, $\boldsymbol{\omega}_i^i$ is angular velocity of Link i and $\boldsymbol{r}_{i,C_i}^i$ is vector from origin of Frame i to eneter of mass C_i.

Next equation (3.11) calculates the angular acceleration of the rotor from Figure 3.2:

$$\boldsymbol{\omega}_{m_i}^{i-1} = \dot{\boldsymbol{\omega}}_{i-1}^{i-1} + k_{ri}\ddot{q}_i\boldsymbol{z}_{m_i}^{i-1} + k_{ri}\dot{q}_i\boldsymbol{\omega}_{i-1}^{i-1} \times \boldsymbol{z}_{m_i}^{i-1}, \tag{3.11}$$

where $\dot{\boldsymbol{\omega}}_{i-1}^{i-1}$ is angular acceleration of the Link $i-1$, k_{ri} is the gear reduction ratio of Motor i, $\ddot{q}_i$ is the general acceleration, $\dot{q}_i$ is the general velocity and $\boldsymbol{z}_{m_i}^{i-1}$ is the unit vector along the rotation vector of Link $i-1$.

Following equation (3.12) compute the force, which act on the Link i:

$$\boldsymbol{f}_i^i = \boldsymbol{R}_{i+1}^i \boldsymbol{f}_{i+1}^{i+1} + m_i\ddot{\boldsymbol{p}}_{Ci}^i, \tag{3.12}$$

where $\boldsymbol{R}_{i+1}^i$ is the rotation matrix from Link i to the Link $i+1$, $\boldsymbol{f}_{i+1}^{i+1}$ is the force which act on the Link $i+1$, m_i is the mass of the Link i and $\ddot{\boldsymbol{p}}_{Ci}^i$ is the linear acceleration of the center of mass of the Link i. As it could be seen, equation (3.12) is equation of the second Newton Law. It could also be seen as the forces effect of the previous Link.

Next equation (3.13) shows the calulation the moment of the Link i:

$$\begin{aligned}\boldsymbol{\mu}_i^i = &-\boldsymbol{f}_i^i \times (\boldsymbol{r}_{i-1,i}^i \times \boldsymbol{r}_{i,C_i}^i) + \boldsymbol{R}_{i+1}^i\boldsymbol{\mu}_{i+1}^{i+1} + \boldsymbol{R}_{i+1}^i\boldsymbol{f}_{i+1}^{i+1} \times \boldsymbol{r}_{i,C_i}^i \\ &+ \overline{\boldsymbol{I}}_i^i\dot{\boldsymbol{\omega}}_i^i + \boldsymbol{\omega}_i^i \times (\overline{\boldsymbol{I}}_i^i\boldsymbol{\omega}_i^i) + \boldsymbol{\omega}_i^i \times (\overline{\boldsymbol{I}}_i^i\boldsymbol{\omega}_i^i) \\ &+ k_{r,i+1}\ddot{q}_{i+1}I_{m_{i+1}}\boldsymbol{z}_{m_{i+1}}^i + k_{r,i+1}\dot{q}_{i+1}I_{m_{i+1}}\boldsymbol{\omega}_i^i \times \boldsymbol{z}_{m_{i+1}}^i\end{aligned}\quad, \tag{3.13}$$

where $\overline{\boldsymbol{I}}_i^i$ is the inertia tensor of Link i. The calculation of inertia tensor is not a part in this work. The values of inertia tensor are gained from CAD program.

Finally the torque can be calculated as:

$$\begin{aligned}\boldsymbol{\tau}_i = {\boldsymbol{\mu}_i^i}^\top {\boldsymbol{R}_i^{i-1}}^\top z_0 + k_{ri} I_{m_i} {\boldsymbol{\omega}_{m_i}^{i-1}}^\top \boldsymbol{z}_{m_i}^{i-1} \\ + F_{vi}\dot{\vartheta}_i + F_{si} sgn(\dot{\vartheta}_i)\end{aligned}, \tag{3.14}$$

where F_{vi} is the viscous friction coefficient and F_{si} is the Coulomb friction coefficient.

As it is said in Siciliano et al. [14], the equations above includes a lot of constants. These constants are $\overline{\boldsymbol{I}}_i^i$, $\boldsymbol{r}_{i,C_i}^i$, $z_{m_i}^{i-1}$ and $z_0 = [0\ 0\ 1]^\top$. The Newton-Euler recursive algorithm has these two phases for given joint space positions, velocities, and accelerations (see in Siciliano et al. [14]):

- With known initial conditions $\boldsymbol{\omega}_0^0$, $\ddot{\boldsymbol{p}}_0^0 - \mathbf{g}_0^0$, and $\dot{\boldsymbol{\omega}}_0^0$, by use (3.7), (3.8), (3.9), (3.10) and (3.11) for $i = 1 \dots N$ can be $\boldsymbol{\omega}_i^i$, $\dot{\boldsymbol{\omega}}_i^i$,$\ddot{\boldsymbol{p}}_i^i$, $\ddot{\boldsymbol{p}}_{Ci}^i$ and $\boldsymbol{\omega}_{m_i}^{i-1}$ computed.

- With known terminal conditions $\boldsymbol{f}_{N+i}^{N+i}$ and $\boldsymbol{\mu}_{N+i}^{N+i}$ use (3.12) and (3.13) for $i = 1 \dots N$ to compute $\boldsymbol{f}_i^i$ and $\boldsymbol{\mu}_i^i$, and then use (3.14) to compute $\boldsymbol{\tau}_i$.

3.5 Studied System

The robotic manipulator for which the algorithm is implemented is KUKA KR5 ARC (see Figure 3.3). Robot KUKA KR5 is six-axes robotic industry manipulator from ARC family which is primarily used for the welding operation. Its maximal payload is up to 5 kilograms.

This thesis aims to the KUKA KR5 of DCE at FEE at CTU in Prague. This robot is connected to the measure card from WAGO company. This card is capable of measuring the power consumption of robot during its movement. The measurement will be used for the comparison of the power consumption of original and optimal trajectory.

3.5.1 Robot Parameters

The parameters of the KR5 were taken from the datasheet and the $machine.dat file. The $machine.dat contains masses of each link and some DH parameters (e.g α). The reason, why these values are stated in the file, is that the robot controller used them for its calculation. The values of some robot parameters are in the following tables 3.1, 3.2, 3.3, 3.4:

Axis	Joint Limits* [°]	Velocity Limits [°/s]	Acceleration Limits [°/s^2]
1	$+50/-40$	+154	+2159.3
2	$+65/-180$	+154	+616
3	$+158/-15$	+228	+1379.3
4	$+350/-350$	+343	+3411.9
5	$+130/-130$	+384	+5229.6
6	$+350/-350$	+721	+5804.9

Table 3.1: Robot parameters - limits of joints

* limits for laboratory of DCE of FEE at CTU in Prague

Axis	Mass [Kg]
1	0
2	19.3
3	26.67
4	7.41
5	2.53
6	0.6

Table 3.2: Robot parameters - masses of links

Axis (Robot)	1	2	3	4	5	6
Axis (Cartesian) [$Kg \cdot m^2$]						
x	0	0.197	0.49	1.7282	0.1046	0.1786
y	0	3.078	3.025	0.509	0.3528	0.2057
z	3.963	1.967	0.799	0.637	0.218	0.027

Table 3.3: Robot parameters - Inertia vectors for each axis

Axis (Robot)	1	2	3	4	5	6
Axis (Cartesian) [m]						
x	0	0.3005	-0.0378	-0.0542	0.0165	0
y	0	0.032	-0.1322	0.0121	0	0
z	0	0	-0.0071	-0.027	-0.0648	0.1333

Table 3.4: Robot parameters - Center of the Mass vectors

The following Figures 3.4 and 3.5 are taken from robot datasheet. The parameters in Figures 3.4 and 3.5 may be not matched with table 3.1 due to workspace limitation of the laboratory.

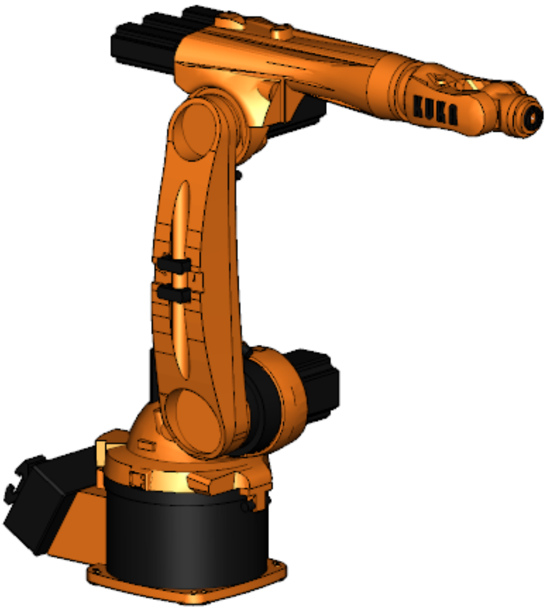

Figure 3.3: KUKA KR5 ARC

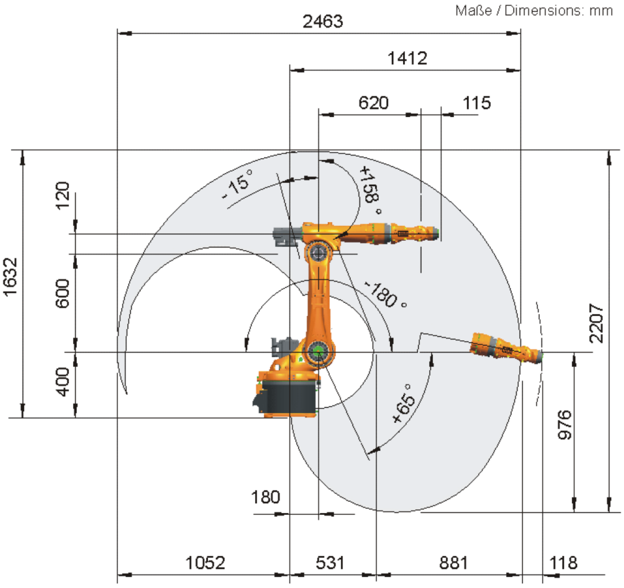
Maße / Dimensions: mm
2463
1412
620
115
120
1632
600
400
- 15°
+158 °
-180 °
+65 °
2207
976
180
1052
531
881
118

Figure 3.4: Operation space of KUKA KR5 - part 1 KUKA Roboter Group GmbH [10]

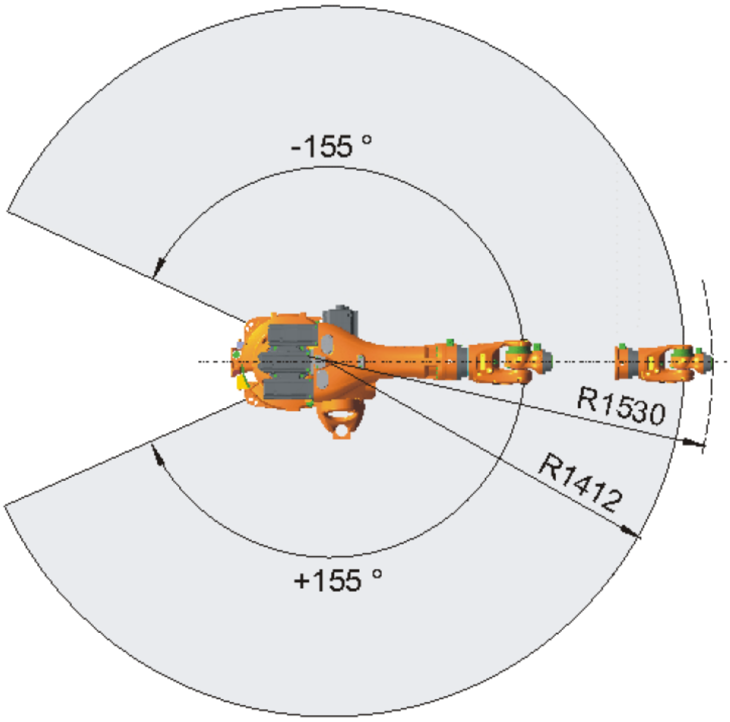
-155 °
+155 °
R1530
R1412

Figure 3.5: Operation space of KUKA KR5 - part 2 KUKA Roboter Group GmbH [10]

Chapter 4

Solving the problem

In this chapter, the algorithm presented in Gleeson et al. [5] is implemented. First the setup of the optimization problem is introduced in section 4.1. Key variables for the problem are presented. In the section 4.2 the criteria function is implemented.

4.1 Problem setup

In this section, the main optimization variables and constraints are introduced.

As it is said in Gleeson et al. [5], the problem could be viewed as following optimal control problem: to find control signal u, that minimize the cost function

$$J = \Phi(x(t_a), t_a, x(t_b), t_b) + \int_{t_a}^{t_b} L(x(t), u(t), t)dt \tag{4.1}$$

while satisfying

$$\dot{x}(t) = f(x(t), u(t), t) \tag{4.2}$$

$$g(x(t), u(t), t) \geq 0 \tag{4.3}$$

$$H(x(t_a), t_a, x(t_b), t_b) = 0 \tag{4.4}$$

for $t \in [t_a, t_b]$,

where $f(x(t), u(t), t)$ is twice differentiable function of states. Function $g(x(t), u(t), t)$ is constraint function and $H(x(t_a), t_a, x(t_b), t_b)$ is Hamiltonian of of optimal control problem.

As it could be seen in Gleeson et al. [5], the cost function is composed of the function Φ, which represents the initial and final cost, and Lagrangian that includes the running cost along the trajectory. This problem will be described in section 4.2.

4.1.1 Optimization variables

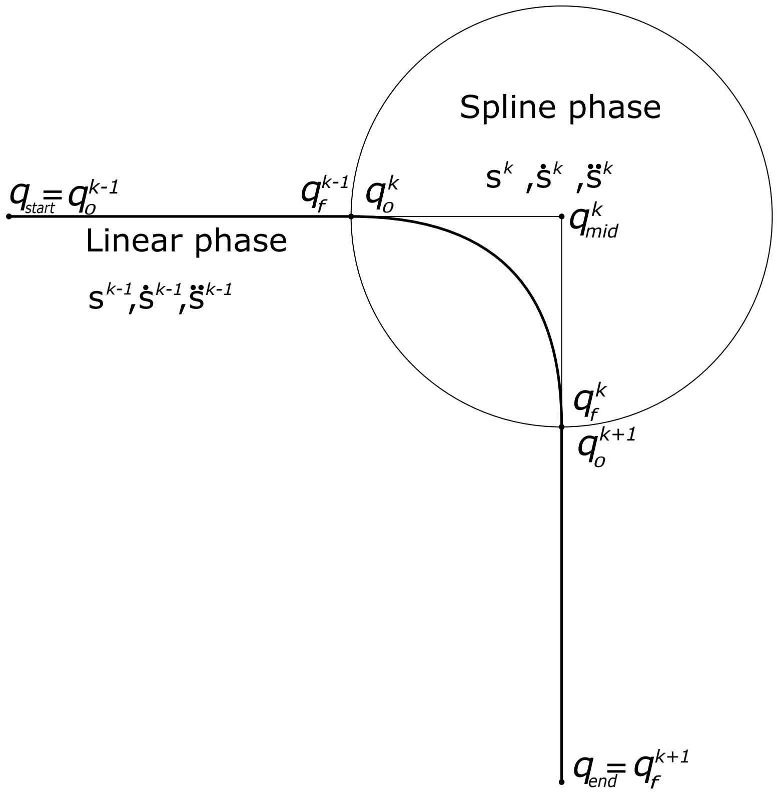

Figure 4.1: Trajectory of the manipulator end-effector.

In the Figure 4.1, a typical robotic trajectory can be seen. The trajectory is set by the via-points, $\boldsymbol{q}_{mid}$, where the direction of the movement is changed. Around these points are areas (zones), where the original trajectory can be optimized. Besides that, the path is also defined by start point $\boldsymbol{q}_{start}$ and the end point $\boldsymbol{q}_{end}$. Areas around the via-points divide the trajectory into two types of phases. Spline phase, which is inside of the zone and linear phase, which connect these zones. Each of these phases has its start point $\boldsymbol{q}_0$, with speed $\dot{\boldsymbol{q}}_0$ and its end point $\boldsymbol{q}_f$ with speed $\dot{\boldsymbol{q}}_f$. These phases also hold its duration in variable Δt and start time t_0. The positions inside the phases are held by parameterized variable s (scalar), which is uniquely spaced in time. The first derivation of this variable s, is parametrized velocity $\dot{s}$ (scalar) and it is introduced for calculation of the velocity. The second derivation is the parametrized acceleration $\ddot{s}$ (scalar), which is used to compute the acceleration. Calculated velocity and acceleration are inputs for calculation of the torque $\boldsymbol{\tau}$, which describes the system dynamics.

The joint space vector is specified by the $\mathcal{P}_q$ degree of freedom of the manipulator. In this thesis, the degree of freedom is six, because in these days the typical manipular used in the manufacture has six axes. The path with $\mathcal{P}_{vp}$ via-points has $\mathcal{P}_{ps}$ spline phases and $\mathcal{P}_{pl}$ linear phases. It can be seen in Figure 4.1, that the total number of phases $\mathcal{P}_c$ can be summarized as:

$$\mathcal{P}_c =_{\mathcal{P}} ps + \mathcal{P}_{pl} = \mathcal{P}_{ps} + \mathcal{P}_{ps} + 1 = \mathcal{P}_{vp} + \mathcal{P}_{vp} + 1 \tag{4.5}$$

Let's assume that the number of discretization points in each phase is N_s. Then the total number of variables is described in Table 4.1.

Varible	Amount
$\boldsymbol{q}$, $\dot{\boldsymbol{q}}$	$N_s\mathcal{P}_c$
$\boldsymbol{\tau}$, $\ddot{\boldsymbol{q}}$, $\ddot{s}$	$(N_s - 1)\mathcal{P}_c$
Δt, t_0	$\mathcal{P}_c$
s, $\dot{s}$	$N_s\mathcal{P}_c$
r	$\mathcal{P}_{sp}$

Table 4.1: Total number of variables concerning robot trajectory

The r in Table 4.1 is zone radius around the via-points, in which the trajectory is optimized. It should be noted that the trajectory in Figure 4.1 is simplified for the better showing of the desired behavior of the algorithm.

4.1.2 Geometrical Optimalization

As mentioned in previous section 4.1.1, the trajectory of robot end-effetor has two types of phases, linear and spline. Both types of phases start with the start point $\boldsymbol{q}_0$ and end with the end point $\boldsymbol{q}_f$. For the finding of these points, the following equations are used:

$$\boldsymbol{v_1} = \mathcal{P}_{vp}^{k} - \mathcal{P}_{vp}^{k-1}, \tag{4.6}$$

$$\boldsymbol{v_2} = \mathcal{P}_{vp}^{k+1} - \mathcal{P}_{vp}^{k}, \tag{4.7}$$

where $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ are vectors directing from the via-point. In this stage of calculation, the start point, and the end point are took as the particular case of via-point, without the zone for optimization. In equations (4.6) and (4.7) the $\mathcal{P}_{vp}^{k}$ represents the via-point k and $\mathcal{P}_{vp}^{k-1}$ is previous via-point and $\mathcal{P}_{vp}^{k+1}$ is the following via-point. After that, using (4.8) and (4.9) are points $\boldsymbol{q}_f^{k-1}$, $\boldsymbol{q}_0^{k}$, $\boldsymbol{q}_f^{k}$ and $\boldsymbol{q}_0^{k+1}$ calculated, where the points denoted by the symbol k are the start point and the end point of the spline phase around the via-point $\mathcal{P}_{vp}^{k}$. Points denoted by the symbol $k-1$ and $k+1$ are the end point and the start point of the adjacent linear phases.

$$\boldsymbol{q}_0^{k} = \boldsymbol{q}_f^{k-1} = \mathcal{P}_{vp}^{k} + d_1\boldsymbol{v}_1, \tag{4.8}$$

$$\boldsymbol{q}_f^{k} = \boldsymbol{q}_0^{k+1} = \mathcal{P}_{vp}^{k} + d_2\boldsymbol{v}_2, \tag{4.9}$$

where d_1 and d_2 are computed like:

$$d_1 = 1 - \frac{r^k}{l_1}, \tag{4.10}$$

$$d_2 = \frac{r^k}{l_2}, \tag{4.11}$$

where r^k is size of the zone around the point $\mathcal{P}_{vp}^k$ in meters. l_1 and l_2 are joint space Euclidean distance between the points $\mathcal{P}_{vp}^k$ and $\mathcal{P}_{vp}^{k-1}$ or between $\mathcal{P}_{vp}^k$ and $\mathcal{P}_{vp}^{k+1}$ for l_2. The distance is calculated:

$$l = \sqrt{\sum_{i=1}^{6} (q_0[i] - q_f[i])^2}, \tag{4.12}$$

where $q_0[i]$ denoted to the i-th part of the position vector $\boldsymbol{q}_0$ respectively $\boldsymbol{q}_f$.

After that, the joint values $\boldsymbol{q}$, between the start point $\boldsymbol{q}_0$ and end point $\boldsymbol{q}_f$, can be calculated with use of the specific parameters s. Parameter s has specific characteristics. By definition (Jönsson and Ustyan [7]):

$$s \in \langle 0, 1 \rangle. \tag{4.13}$$

For linear phase, it is possible to calculate points $\boldsymbol{q}$ by linear interpolation:

$$\boldsymbol{q} = \boldsymbol{q}_0 + s(\boldsymbol{q}_f - \boldsymbol{q}_0). \tag{4.14}$$

For the spline phase, the path of end-effector is specified by the parameter s as in the case of linear phase. But the path within the zone is determined by three points $\boldsymbol{q}_0$, $\boldsymbol{q}_f$, $\boldsymbol{q}_{mid}$. For calculations of the path joint points $\boldsymbol{q}$, the following polynomial expression is used:

$$\boldsymbol{q} = \tilde{q}_1(s) + p(s)(\tilde{q}_2(s) - \tilde{q}_1(s)), \tag{4.15}$$

where $\tilde{q}_1(s)$ and $\tilde{q}_2(s)$ are linear interpolation between the points $\boldsymbol{q}_0$, $\boldsymbol{q}_f$, $\boldsymbol{q}_{mid}$.

$$\tilde{q}_1(s) = \boldsymbol{q}_0 + s(\boldsymbol{q}_{mid} - \boldsymbol{q}_0), \tag{4.16}$$

$$\tilde{q}_2(s) = \boldsymbol{q}_{mid} + s(\boldsymbol{q}_f - \boldsymbol{q}_{mid}). \tag{4.17}$$

The polynomials $p(s)$ in equation (4.15) are chosen due to its smoothness property. As it is said in Gleeson et al. [5], a smooth transition at $s = 0$ and $s = 1$ with the continuous derivative of order m requires a polynomial $p(s)$, of degree $2m + 1$. First, second and third order of continuity at the phase boundaries has been implemented which corresponds to the following three polynomials (Gleeson et al. [5]).

$$p_3(s) = 3s^2 - 2s^3 \tag{4.18}$$

$$p_5(s) = 10s^3 - 15s^4 + 6s^5 \tag{4.19}$$

$$p_7(s) = 35s^4 - 84s^5 + 70s^6 - 20s^7 \tag{4.20}$$

The following Figure 4.2 shows, how these polynomials behave from $s = 0$ to $s = 1$.

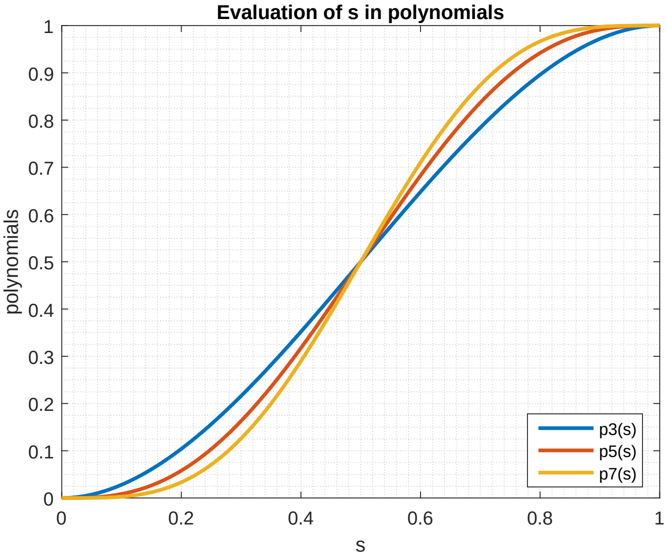

Figure 4.2: Evaluation of s in polynomials (4.18), (4.19) and (4.20)

The equations (4.14) and (4.15) are taken from Gleeson et al. [5]. It can be seen that the equation (4.14) looks like the special case of equation (4.15). It is possible to get equation (4.14) from (4.15) by setting the polynomial equal to zero and replacing the variable $\boldsymbol{q}_{mid}$ with the variable $\boldsymbol{q}_f$. Then it is possible to treat each phase with one equation only and the linear phase is the particular case of spline phase. The final equation in the application is:

$$\boldsymbol{q} = \tilde{q}_1(s) + p(s)(\tilde{q}_2(s) - \tilde{q}_1(s)) \begin{cases} \text{like in (4.10) for spline phase,} \\ \boldsymbol{q}_{mid} = \boldsymbol{q}_f, \quad p(s) = 0 \text{ for linear phase} \end{cases} \tag{4.21}$$

Values calculated by the equation (4.21) have to obey maximal and minimal boundaries value for joint space position.

$$\boldsymbol{q}_{lower} \leq \boldsymbol{q} \leq \boldsymbol{q}_{upper}$$

These limitations are set by the workspace of the robot.

4.1.3 Kinematic Optimization

To be sure, that the variables presented in the previous section (4.1.1) optimized the trajectory geometrically and kinematically, the following constraints have to be implemented. In paper Gleeson et al. [5], there are these constraints:

$$-s_i + 2s_{mp_i} - s_{i+1} = 0, \tag{4.22}$$

$$-\dot{s}_i + 2\dot{s}_{mp_i} - \dot{s}_{i+1} = 0, \tag{4.23}$$

$$s_{i+1} - s_i - \dot{s}_{mp_i}\Delta t/N = 0, \tag{4.24}$$

$$\dot{s}_{i+1} - \dot{s}_i - \ddot{s}_i\Delta t/N = 0, \tag{4.25}$$

where s_i is parameter s at discretization step i in one phase and s_{i+1} is parameter s at discretization step $i+1$ in same phase.

Equations (4.22) and (4.23) represent the midpoint constraints, which is used when the midpoint method for discretization of the criteria function is used. Equations (4.24) and (4.25) are used for derivation constraints, which connect the first and second time derivation of the parameter s presented in section 4.1.1. The equations presented here can be used both for linear phase and for the spline phase because the parameter s is independent of the type of the phases.

However, the implementation of the midpoint discretization method brings another constraint into the optimization problem, which has to be fulfilled. The equation (4.25) looks like the forward Euler discretization method, and the decision for using it was made. The constraints, which make sure, that the time derivation of parameter s fulfills the time evolution are:

$$s_{i+1} - s_i - \dot{s}_i \Delta t/N = 0, \tag{4.26}$$

$$\dot{s}_{i+1} - \dot{s}_i - \ddot{s}_i \Delta t/N = 0. \tag{4.27}$$

Optimalization of Speed

The first time derivation of parameter s is used for the calculation of optimal velocity. The velocity in the joint space $\dot{\boldsymbol{q}}$ calculated with parameter $\dot{s}$ has fulfilled the following constraints for the linear phase:

$$\dot{\boldsymbol{q}}_{lower} \leq \dot{s}_i(\boldsymbol{q}_f - \boldsymbol{q}_0)/l \leq \dot{\boldsymbol{q}}_{upper}, \tag{4.28}$$

where l is calculated by the equation (4.12).

The velocity in the spline phase is computed by the time derivation of the equation (4.15), which has to fulfill the constraints $\dot{\boldsymbol{q}}_{lower} \leq \dot{\boldsymbol{q}} \leq \dot{\boldsymbol{q}}_{upper}$:

$$\dot{\boldsymbol{q}} = \dot{s}(\boldsymbol{q}_{mid} - \boldsymbol{q}_0) + \dot{s}\frac{dp(s)}{ds}(\tilde{q}_2(s) - \tilde{q}_1(s)) + \dot{s}p(s)(\boldsymbol{q}_f - \boldsymbol{q}_{mid} + \boldsymbol{q}_0). \tag{4.29}$$

Like in the previous section, the linear phase is the special case for spline phase. The final equation used in the application is:

$$\begin{aligned} \dot{\boldsymbol{q}} =& \dot{s}(\boldsymbol{q}_{mid} - \boldsymbol{q}_0) + \dot{s}\frac{dp(s)}{ds}(\tilde{q}_2(s) - \tilde{q}_1(s)) + \\ &+ \dot{s}p(s)(\boldsymbol{q}_f - \boldsymbol{q}_{mid} + \boldsymbol{q}_0) \begin{cases} \text{like in (4.29) for spline phase,} \\ \boldsymbol{q}_{mid} = \boldsymbol{q}_f, \;\; p(s) = 0 \;\; \text{for linear phase} \end{cases} \end{aligned} \tag{4.30}$$

The velocity at the phases boundaries has to be equal. The parametric variable $\dot{s}$ is not only limited by the maximum or minimum joint space velocity but also by the following constraints:

$$\dot{\boldsymbol{q}}_f^{k-1} - \dot{\boldsymbol{q}}_0^k = 0, \tag{4.31}$$

where $\dot{\boldsymbol{q}}_f^{k-1}$ is final velocity of the phase $k-1$ and $\dot{\boldsymbol{q}}_0^k$ is initial velocity of the phase k.

$$\dot{\boldsymbol{q}}_f^k - \dot{\boldsymbol{q}}_0^{k+1} = 0, \tag{4.32}$$

where $\dot{\boldsymbol{q}}_f^k$ is final velocity of the phase k and $\dot{\boldsymbol{q}}_0^{k+1}$ is initial velocity of the phase $k+1$. With both of these constraints, it is ensured that the velocity at the phase boundaries has the same direction and the absolute value of velocity is same at both side of boundaries.

Optimalization of Acceleration

For the calculation of the acceleration, the second derivation of parameter s is used. The acceleration for the linear phase is according to Gleeson et al. [5] is calculated and it is constraint by:

$$\ddot{\boldsymbol{q}}_{lower} \leq \ddot{s}_i(\boldsymbol{q}_f - \boldsymbol{q}_0)/l \leq \ddot{\boldsymbol{q}}_{upper} \tag{4.33}$$

The acceleration for the spline phase is calculated like the second time derivation of equation (4.15). The computed acceleration also has to fulfill the constraints, which are the limitations of the maximal robot acceleration.

$$\begin{aligned} \ddot{\boldsymbol{q}} =& \ddot{s}(\boldsymbol{q}_{mid} - \boldsymbol{q}_0) + \left(\ddot{s}\frac{dp(s)}{dt} + \dot{s}^2\frac{d^2}{ds^2}p(s)\right)(\tilde{q}_2(s) - \tilde{q}_1(s)) + \\ &+ \left(2\dot{s}^2\frac{d}{ds}p(s) + \ddot{s}p(s)\right)(\boldsymbol{q}_f - \boldsymbol{q}_{mid} + \boldsymbol{q}_0) \end{aligned} \tag{4.34}$$

The final equation used in application is same as equation (4.34) for spline phase. For linear phase the variables $\boldsymbol{q}_{mid} = \boldsymbol{q}_f$ and polynomial $p(s) = 0$.

4.2 Implemation of the Criterion function

The joint space parameters calculated in the previous section are then used for evaluation of the criterion function. First, the torque, which acts on the robot during the movement is computed. The torque is then used for the calculation of negative electric power, which is generated by the torques and the forces, which are caused by the movement.

4.2.1 Calculation of the torques

For the calculation of the torques, the Mathematical model of the manipulator has to be implemented. As it is written in the previous section 3.4.2, the Daneavit-Hartenberg parameters have to be found. If the steps one to four presented in section 3.3.1 are followed, the following parameters are found:

Link i	α_i [rad]	d_i [m]	a_i [m]
1	$\pi/2$	0.400	0.180
2	0	0.135	0.600
3	$\pi/2$	0.135	0.120
4	$\pi/2$	0.620	0
5	$\pi/2$	0	0
6	0	0.115	0

Table 4.2: Found Daneavit-Hartenberg parameters

Parameter θ of DH parameters is replaced by the angle of rotation q computed by equation (4.21). Parameters a and d were found in the datasheet or $machine.dat of KUKA KR5.

The equations of the mathematical model used in the final application are calculated by the ReDySim algorithm by Shah [13]. ReDySim (Recursive Dynamics Simulator) is free-to-use sets of Matlab scripts, which compute the inverse and forward dynamic recursive equations of the robots and multibody systems. Parameters necessary for the calculation of the inverse dynamic are input into the ReDySim algorithm. Redysim already contains inbuilt parametrization of KR5. Inverse dynamic symbolic equations for each Link are output. The only parameters which are not filled into the equations are parameters for joint space position, velocity, and acceleration. These parameters are supplied by the solver during the optimization process.

4.2.2 Calculation of the currents

The torque computed in the previous section is then applied to the calculation of the electric current, which is generated in the motor of the robot by the movement of the Link mass. First, the current has to be calculated. In this thesis, the complex system of the synchronous motor is replaced by approximation of DC motor. The model of DC motor is described by the bond graph in Figure 4.3. This model is used for each motor separately.

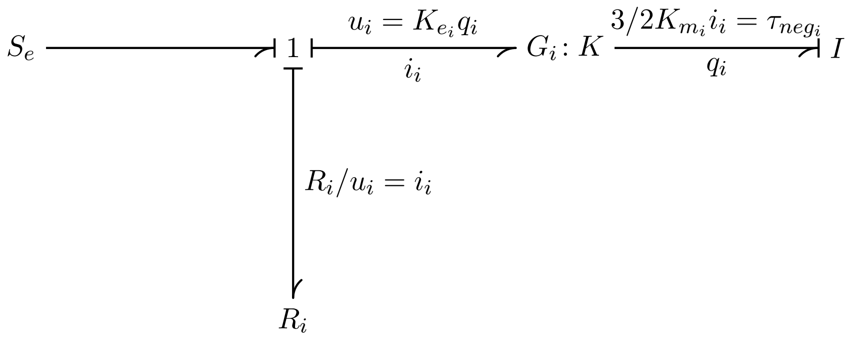

Figure 4.3: Bond graph of motor i

The constants K in Figure 4.3 are computed as follows:

$$K_{e_i} = K_{u_i} \frac{60 \times \sqrt{2}}{1000 \times 2\pi\sqrt{3}}. \tag{4.35}$$

The constant K_{e_i} is a current constant used for the conversion of the joint space velocity into the voltage. Constant K_{u_i} is a voltage constant of the unit $V/1000ot \times min^{-1}$ Souček [15]. The value of K_{u_i} for each motor, could be found in Table 4.3 for the tested robot.

$$K_{m_i} = K_{\tau_i} \frac{2}{3\sqrt{2}}. \tag{4.36}$$

K_τ represents the moment constant, which is used for the calculation of the force constant. The force constant K_m serves then for the computation of the negative torque by multiplying with current of motor. The negative torque describes the torque caused by the movement of the mass of the Link.

Link i	Manufacturer	Model	K_u [$V/1000ot \times min^{-1}$]	K_τ [Nm/A]	R [Ω]
1, 2	Siemens	1FK7080	105	1.61	0.98
3	Siemens	1FK7042	89	1.40	4.67
4, 5, 6	Siemens	1FK7032	45	0.67	5.05

Link i	Manufacturer	Model	K_m [N_m/A]	K_e [$V/1000ot \times min^{-1}$]
1, 2	Siemens	1FK7080	0.7590	0.8187
3	Siemens	1FK7042	0.6600	0.6930
4, 5, 6	Siemens	1FK7032	0.3158	0.3509

Table 4.3: Parameters of robot motor

It has to be noted that this modeling of the motor is simplified. The equations for calculation of constants K are more explained in Souček [15].

4.2.3 Evalution of the Criterion function

The currents which are in the motor (for each motor separately) has to be introduced:

$$\boldsymbol{i}_c = \boldsymbol{i}_{neg} + \boldsymbol{i}_p. \tag{4.37}$$

In equation (4.37), $\boldsymbol{i}_c$ represents the overall current consumed inside the motor during the movement. Variable $\boldsymbol{i}_p$ is the positive current, which came to the motor from an electrical network. Variable $\boldsymbol{i}_{neg}$ represents the negative current, presented in the previous section. This current is caused by the movement of the mass of Link. It is produced inside the motor. Simplified goal of the criterial function is to minimize the difference between positive and negative current.

For following equations next definition is used:

$$k_m = \frac{3}{2} K_m. \tag{4.38}$$

Variable k_m on left side of equation (4.38) is from equation (4.36). This definition is made for the better readability in equations.

The overall torque can be calculated as the sum of the positive torque (caused by positive current) and the negative torque (caused by negative torque). All $\boldsymbol{\tau}$ and $\boldsymbol{i}$ are funtions of time:

$$\boldsymbol{\tau} = \boldsymbol{\tau}_p + \boldsymbol{\tau}_{neg} \tag{4.39}$$

or as it was shown in section 4.2.2, overall torque can be calculated like:

$$\boldsymbol{\tau} = k_m \boldsymbol{i}_c = k_m \boldsymbol{i}_p + k_m \boldsymbol{i}_{neg} \tag{4.40}$$

and if the equation (4.40) is expressed as:

$$k_m \boldsymbol{i}_p = \boldsymbol{\tau} - k_m \boldsymbol{i}_{neg} = \boldsymbol{\tau} - \boldsymbol{\tau}_{neg} = \boldsymbol{\tau}_p. \tag{4.41}$$

To calculate the energy caused by the currents, the power in each discretization step i has to be computed. Power at one discretization step i in phase k can be calculated like:

$$P^{k,i} = \dot{\boldsymbol{q}}^{i\top} \boldsymbol{\tau}^i. \tag{4.42}$$

The equation (4.42) came into following form, when the equation (4.39) is used:

$$P^{k,i} = \dot{\boldsymbol{q}}^{i\top} (\boldsymbol{\tau}_p^i + \boldsymbol{\tau}_{neg}^i). \tag{4.43}$$

The positive torque can be negative because of used method of calculation of the joint space velocity. The industrial manipulators don't have the capability of the recuperation of the energy. The recuperation is a process of returning the energy back into the electrical network, when the robot is decelerating or stopping. Therefore the energy, which comes into the system from the network has to be consumed, and equation (4.43) has to go over into next form:

$$P^{k,i} = |\dot{\boldsymbol{q}}^{i\top} \boldsymbol{\tau}_p^i| + \dot{\boldsymbol{q}}^{i\top} \boldsymbol{\tau}_{neg}^i \tag{4.44}$$

and for each link in phase k and for each discretization step i:

$$P^k = \sum_{i=1}^{N_s-1} \left(\sum_{j=1}^{N_{axis}} \left(|\dot{q}_j^i \tau_{p_j}^i| + \dot{q}_j^i \tau_{neg_j}^i \right) \right),$$

$$\dot{\boldsymbol{q}}^i = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_{N_{axis}-1} \\ \dot{q}_{N_{axis}} \end{bmatrix}, \; \boldsymbol{\tau}^i = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_{N_{axis}-1} \\ \tau_{N_{axis}} \end{bmatrix}. \tag{4.45}$$

The reason why the power at last point of phase k is not computed is that the last point is same as the first point of the following phase $k+1$. This step ensures that the power consumption is not calculated twice. For the last phase in trajectory, the last point is of course computed, so i goes to N_s.

If into the equation above the equation (4.41) is infilled, it came into:

$$
\begin{gathered}
P^k = \sum_{i=1}^{N_s-1} \left(\sum_{j=1}^{N_{axis}} \left(|\dot{q}_j^i(\tau_j^i - \tau_{neg_j}^i)| + \dot{q}_j^i \tau_{neg_j}^i \right) \right), \\
\dot{\boldsymbol{q}}^i = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \vdots \\ \dot{q}_{N_{axis}-1} \\ \dot{q}_{N_{axis}} \end{bmatrix}, \ \boldsymbol{\tau}^i = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \vdots \\ \tau_{N_{axis}-1} \\ \tau_{N_{axis}} \end{bmatrix}.
\end{gathered}
\tag{4.46}
$$

With the equation (4.46), it is possible to calculate the power acting in phase k. To calculated the energy, the result form (4.46) has to be multiplied by the discretization constant h^k in phase k, which is computed like:

$$
h^k = \frac{\Delta t^k}{N_s}, \tag{4.47}
$$

where Δt^k is time duration of phase k. Noted that the h^k is equal whitin the phase. The final form of criteria function is:

$$
\begin{aligned}
J &= \sum_{k=1}^{N_p} h^k P^k \\
&= \sum_{k=1}^{N_p} \left(h^k \sum_{i=1}^{N_s-1} \left(\sum_{j=1}^{N_{axis}} \left(|\dot{q}_j^i(\tau_j^i - \tau_{neg_j}^i)| + \dot{q}_j^i \tau_{neg_j}^i \right) \right) \right),
\end{aligned}
\tag{4.48}
$$

where N_p is number of phases.

Equation (4.48) is a discretization form of the criteria function presented in equation (4.1) for energy optimization. As it is said in Gleeson et al. [5], the function could be modified for the different purpose of optimization.

Chapter 5

Experiments

The equations and algorithm presented in the previous chapter are implemented in the application in Matlab programming language. The test manipulator, the KUKA KR5 ARC of the department of control engineering of FEE at CTU in Prague was used.

Before the test on the real manipulator, the optimal trajectory has to be computed by implemented application. The output of the application is offline made robotic code (program) in KRL (Kuka robotic language), which can be immediately used on the real manipulator. Two trajectories are chosen for a proof of concept. First one is the simple trajectory consisting of three points. The second one is more complex trajectory with five points. All joints are moving in the second trajectory.

5.1 Experiment in Simulated Environment

Points of original trajectory are the input for the program. These points are used by equations (4.8) and (4.9) for calculation of start and end points of each phase. The calculation also needs the radius in each via-point of the trajectory. The part of the trajectory inside of the spehere with this radius is treated as a spline phase. Another important input variable is a number of points for discretization of each phase and the array with parameter specifying the type of phase. The variable 0 denotes linear phase and variable 1 denotes spline phase. It is possible that the spline phase follows the spline phase. In further, it is possible to have more linear or spline phases right after each other. Each phase has its own sampling frequency obtained during optimization. Therefore, the Figures 5.5, 5.6, 5.7, 5.8, 5.11,5.12 , 5.13 and 5.14 on the x axis have a discretization step instead of time.

5.1.1 Original Trajectory

The variables mentioned above are held in the main script *main.m* of application. In this script, it is possible to set the name of output robotic program and location, where the program will be saved. Optimization process is started by the execution of the scripts. The parameters for original trajectory are computed first. In the original trajectory, all phases are linear (in carthesian space) and each start and end points are „fine“. This means that the robot has to stop in it. Because the number of phases in original trajectory is smaller than in optimized one, the number of discretization variables is larger. The number of discretization points in original trajectory is increased or decreased dependence on a number of discretization points in optimal one so the number of discretization points is equal.

The positions are calculated by equation (4.14). The velocities are calculated by the time which is computed by standard trapezoidal motion profile with equal and constant acceleration and deceleration Stienecker [17]. The time is computed like:

$$\Delta t^k = \frac{\Delta\theta_i}{v_{max_i}} + \frac{v_i^k}{a_{max_i}} \iff \frac{v_{max_i}}{a_{max_i}} < \Delta\theta_i, \tag{5.1}$$

or

$$\Delta t^k = 2\sqrt{\frac{\Delta\theta_i}{a_{max_i}}} \iff \frac{v_{max_i}}{a_{max_i}} \geq \Delta\theta_i, \tag{5.2}$$

where Δt^k is duration of time in phase k. The variables v_{max_i} and a_{max_i} are absolute maximal speed and acceleration vectors (see Table 3.1) in joint i. The variable $\Delta\theta_i$ denotes the absolute joint-space trajectory length in joint i.

The acceleration is then calculated like a derivation of the velocity by Matlab function *diff()*. The variables are inserted into the equation (4.46). The result is in next used for calculation of energy (equation (4.48)) by multiplying with time from equation (5.1) or (5.2).

5.1.2 Optimal Trajectory

The next step in the application is an optimization. Phases start and end points, which were created by equations (4.8) and (4.9) are inserted and held in an object of class *energyModel*. The objects of class *energyModel* have implemented all necessary equations presented in chapter 4. The equations (4.30) and modified version (4.34) are created by Matlab function *diff()* as a first and second derivation of equation (4.21). The classes of phases are right after that inserted into the Matlab anonymous function, for more information about anonymous function see MathWorks [11], which can be considered as criteria function of the optimization problem. Then the nonlinear solver is set up. Chosen solver is IPOPT solver presented in section 3.1. The solver tries to find out the set of variable s, $\dot{s}$ and $\ddot{s}$. These variables are held in one array (vector) because the IPOPT solver in Matlab does not support multiple inputs variables. After the all solver options are set up, the optimization started.

Originaly the trajectories consisting of at most three phases were optimized in one step. The optimization of the trajectory which has seven phases took one and half hour to solve. The time of optimization grows almost exponentially. The reason why the time of optimization took a long time is a number of control variables. For example, if there are five phases and in each phase are five discretization points the total number of control variables are seventy-five. Because of that, the decision to separate the trajectories into the smaller pieces was made. This decision caused that the optimal solution of optimization is suboptimal. For the trajectories which have more than three phases following algorithm is used:

```
while satisfying all constraints;
for $i$ = 1 to $k$ with step 2 do
    $[M^i, \dots, M^{i+2}] = \text{optimize}([M^i, \dots, M^{i+2}])$;
end
```

Algorithm 1: Used suboptimal optimization strategy.

where $\boldsymbol{M}$ is array of phases and variable k is total number of phases. To ensure, that the phases between the suboptimal solution match, the same constraints as (4.31) and (4.32) are used, but for the suboptimal boundaries. Using these steps, the „suboptimal“ trajectory is found. After that, the optimal trajectory is transferred into KRL code which can be later executed on real KUKA manipulator.

5.1.3 Results from simulation

In this section, the results of the simulation are presented. As it was said, two types of trajectories described in section 5.1 were chosen. Firstly the results for easiest one will be introduced.

First Trajectory - Results

The first trajectory has three points: the start point, the end point, and one via-point. At this via-point the size of the radius was changed gradually. Following table show how the change of radius modify the difference between positive and negative energy presented in previous chapter 4:

Radius $[m]$	Final Time $[s]$	Final Energy $[J]$	Energy Save [%]	Time Save [%]
fine	1.16	364.73	0	0
0.05	1.16	153.35	57.96	0
0.1	1.16	155.25	57.43	0
0.15	1.16	147.86	59.46	0
0.2	1.16	175.66	51.84	0
0.25	1.16	188.45	48.33	0
0.3	1.16	136.49	62.58	0
0.35	1.16	138.67	61.98	0

Table 5.1: Energy save for simulation - Trajectory 1

And table 5.2 shows distance between the points in Cartesian space:

	$V_1 \rightarrow V_2$	$V_2 \rightarrow V_3$
Distance $[m]$	0.552	0.516

Table 5.2: Distance between points of original Trajectory 1 (see Figure 5.1)

As it is seen in Table 5.1, the best optimization was 62.58 % of the original one („fine“). These energy reductions are partly because the robot does not stop in the via-points as in original trajectory. The final time of each new trajectories in Table 5.1 is same as in original one. It is because the criterial function is primarily written for energy saving. For the solution of the problem was better to fulfill the given upper time boundary (original duration). Even the experiments with longer time boundary then the original one, leads to the fulfillment of the set time limits. That could be caused by acceleration because the torque is associated with acceleration.

The application produces many graphs. The most important ones will be shown here. The rest of figures could be seen in Appendix B.

Figures 5.1 and 5.2 show the trajectory before optimization and after. The zone radius is 0.2 meters. The trajectory is smooth after the optimization. Note that the trajectory looks linear only between the chosen joints and only in joint space. The Figure 5.3 shows the trajectory in Cartesian space. In the Figure 5.3 the behavior of PTP motion is shown. The Figure 5.4 shows the front view of the trajectory. The blue line is the original trajectory, and the red line is the optimized one. Both pictures were taken in Process Simulate. Process Simulate is process software by Siemens for simulation and verification of industrial process with many useful functions. One of them is an easy visualization of robots movements.

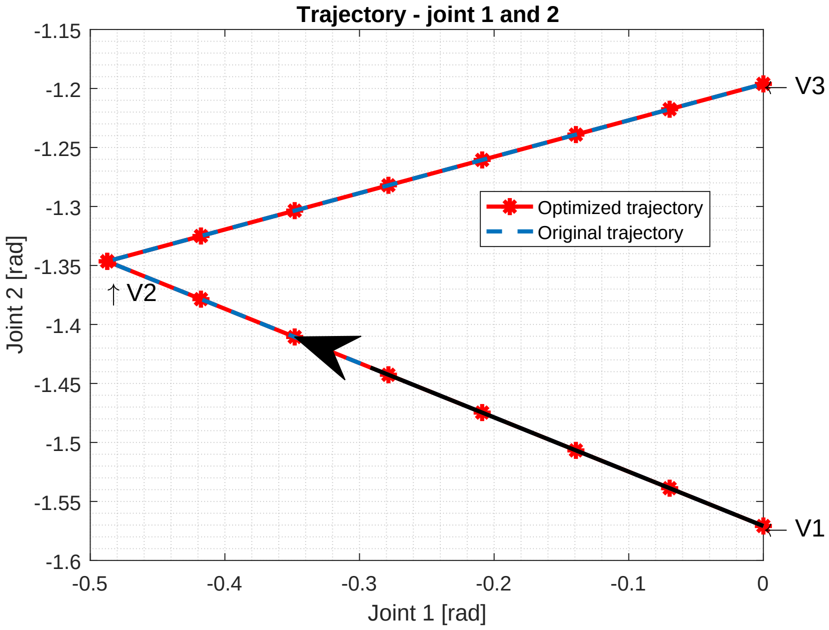

Figure 5.1: Computed position of original Trajectory 1. The black arrow shows direction of the movement.

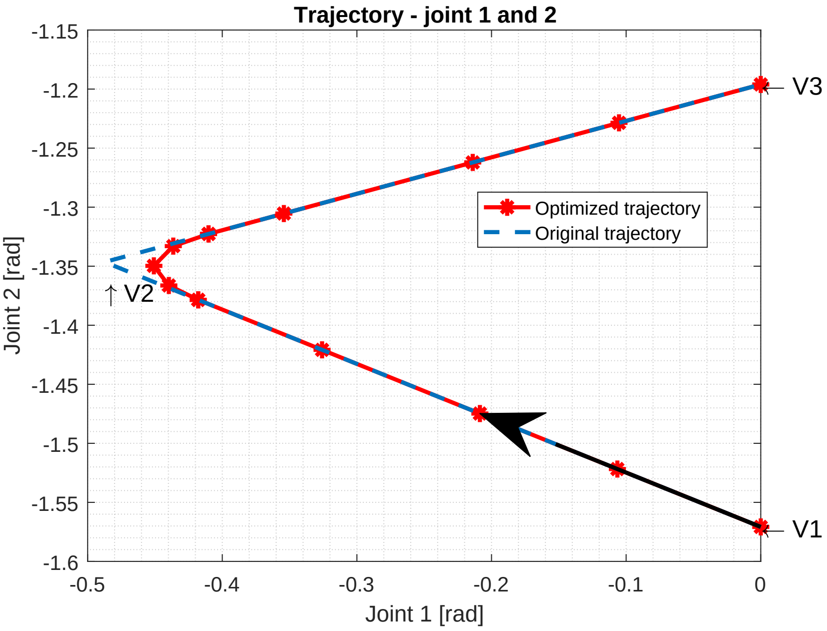

Figure 5.2: Computed position of optimazed Trajectory 1. The black arrow shows direction of the movement.

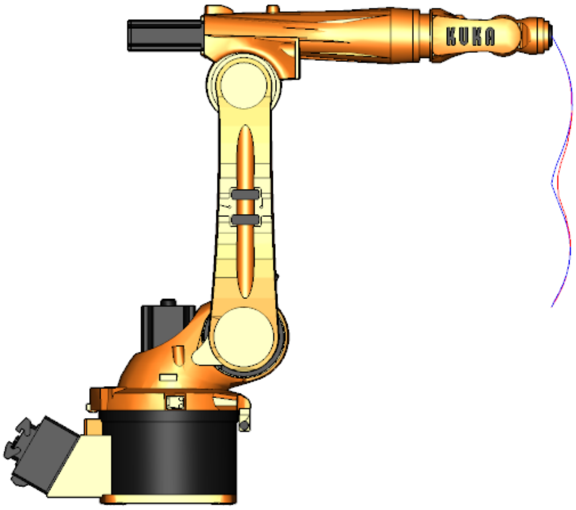

Figure 5.3: Trajectory 1 - side view from Process Simulate

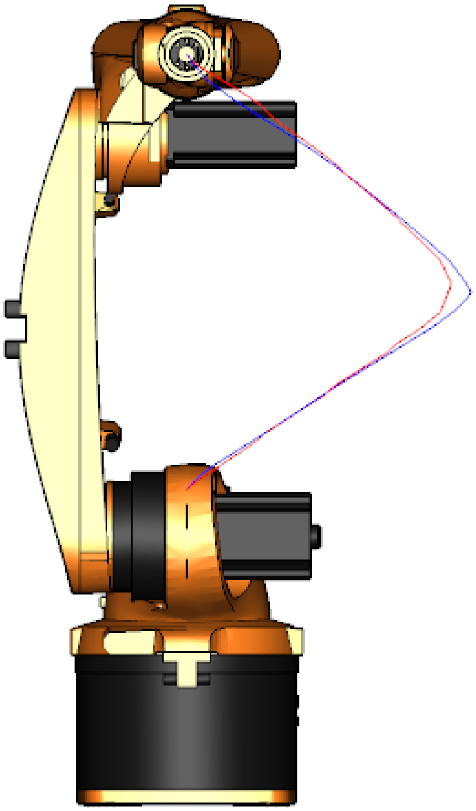

Figure 5.4: Trajectory 1 - front view from Process Simulate

Figures 5.5 and 5.6 show the velocity and acceleration of the first joint. The Figure 5.5 has standard trapezoidal motion profile. The Figure 5.6 presents the slowing down the maximum speed by optimization tool. The force is therefore minimized. The closer look shows how the constraints (4.31) and (4.32) adjust the speed and acceleration at the same position (ex. samples 4 and 5). The acceleration at the last point of phase $k-1$ (sample 4) is equal to zero, but speed is same as the speed at the first point of phase k (sample 5), which has nonzero acceleration. As it was mentioned in the explanation of (4.45) the last point N (sample 4) of phase $k-1$ is neglected. It has to be also noted that the initial and final velocity of the trajectory could not be set. One of the reasons is that the solver finds these velocities, which are best for achieving the minimal energy consumption.

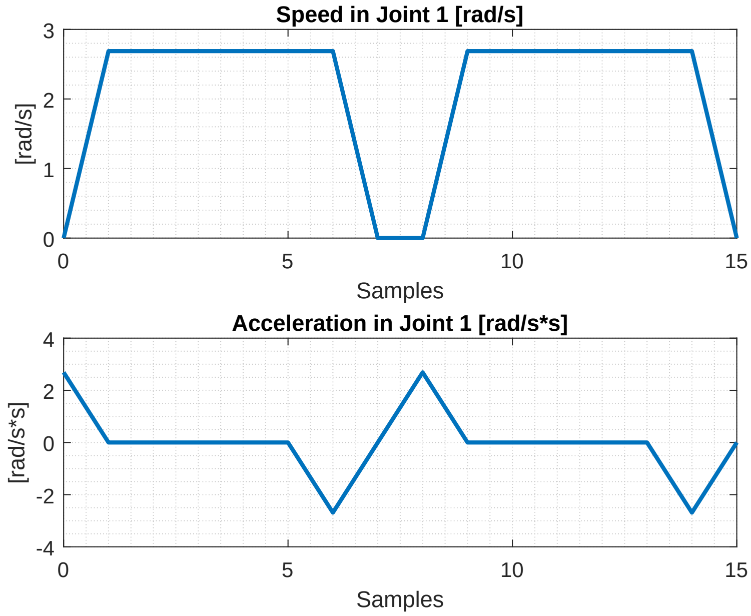

Figure 5.5: Close look up at speed and acceleration of first joint - original trajectory

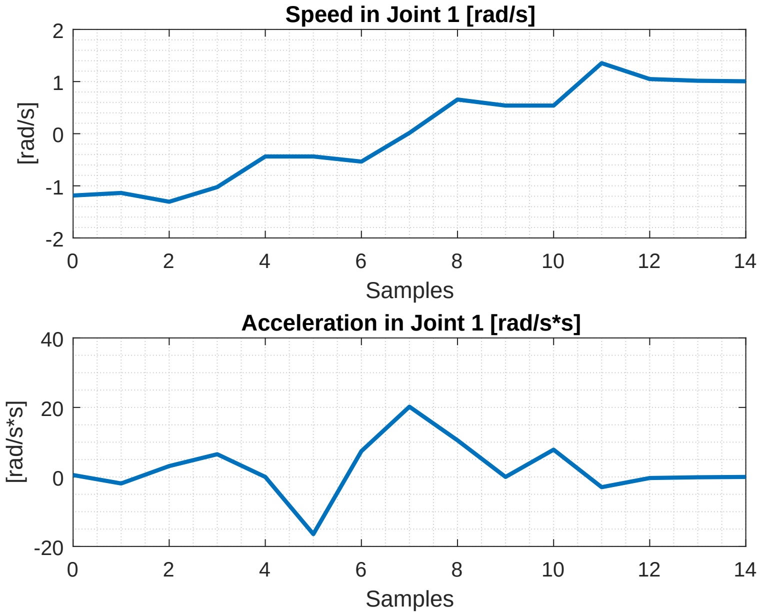

Figure 5.6: Close look up at speed and acceleration of first joint - Optimal one

Figures 5.7 and 5.8 present the power consumption of the manipulator before and after the optimization. From the figures, the times when the manipulator accelerated and decelerated can be seen. The curve in Figure 5.7 (orginal one) is smoother than in the Figure 5.8. However, the maximum power of optimized trajectory is noticeably lower.

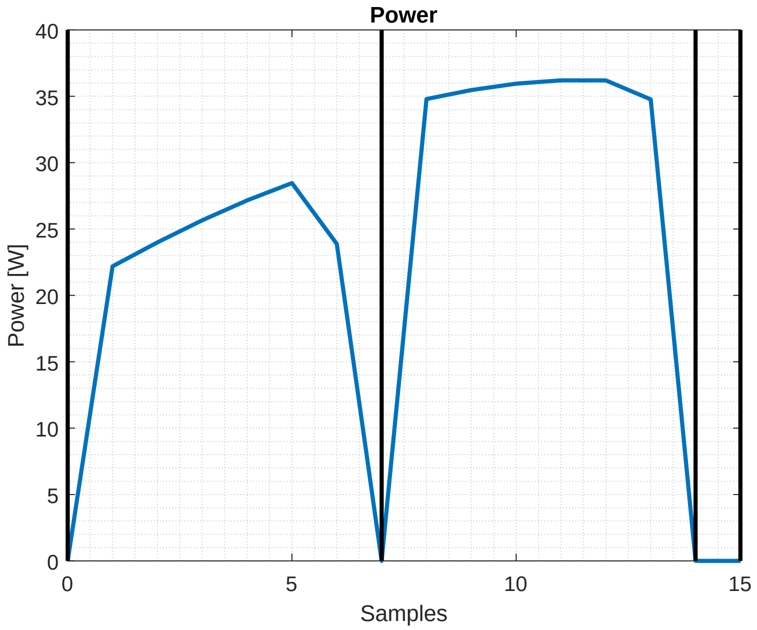

Figure 5.7: Computed power in each discretization step - original trajectory

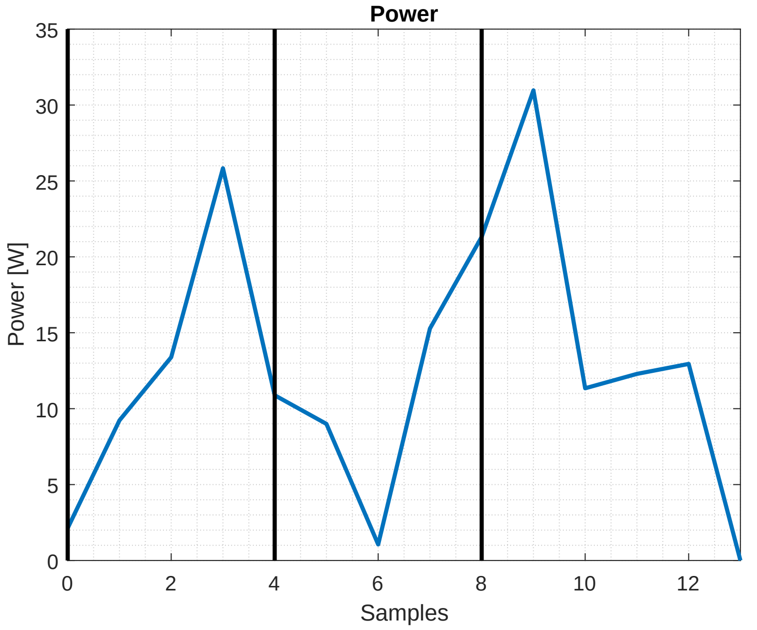

Figure 5.8: Computed power in each discretization step

Second Trajectory - Results

The second trajectory (Figure 5.9) is more complicated than the previous one. It has five points through which the end-effector has to go. Overall it has three via-points (points with zone radius for optimization). In this trajectory, it is guaranteed that each joint is moving during the movement. For optimization process, all via-points has the same radius. The following Table 5.3 shows the results from optimization:

Radius $[m]$	Final Time $[s]$	Final Energy $[J]$	Energy Save [%]	Time Save [%]
fine	2.74	630.9	0	0
0.05	2.74	292.28	53.67	0
0.1	2.74	292.14	53.69	0

Table 5.3: Energy save for simulation - Trajectory 2

	$V_1 \rightarrow V_2$	$V_2 \rightarrow V_3$	$V_3 \rightarrow V_4$	$V_4 \rightarrow V_5$
Distance $[m]$	0.55	1.2	0.7	0.54

Table 5.4: Distance between points of Trajectory 2 (see Figure 5.9)

Figures 5.9 and 5.10 show how trajectory looks before optimization and after it. In the figure, 5.10 can be seen where trajectory starts and end. The radius at via-points is set to 0.1 meters. At Figure 5.10 can be seen, that the optimization zone isn't the same at all via-points. The zone in joint space is not same as the zone in the Cartesian space, see 3.3.1.

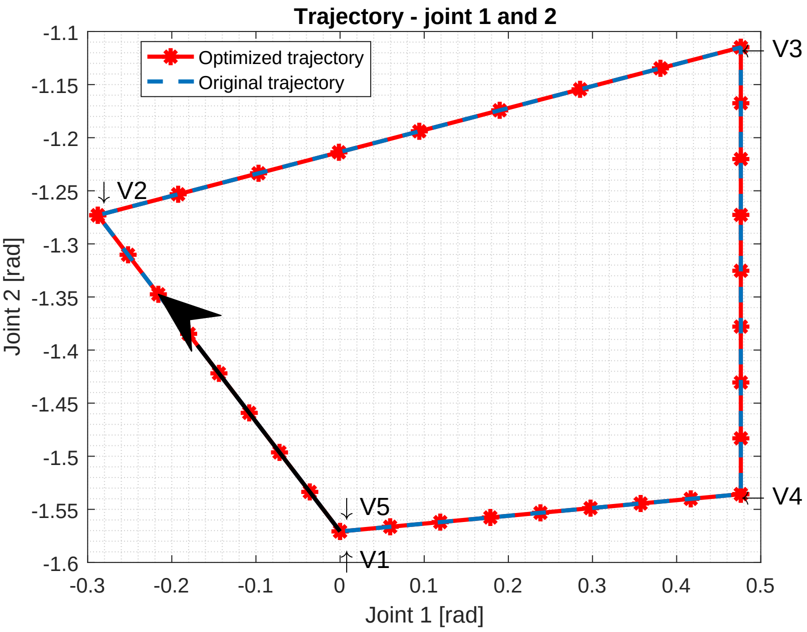

Figure 5.9: Computed position of original Trajectory 2. The black arrow shows direction of the movement.

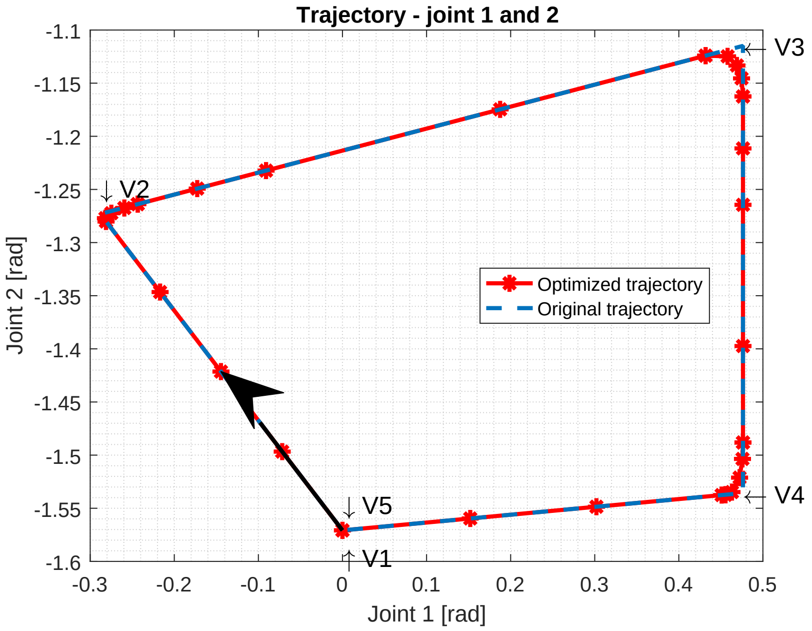

Figure 5.10: Computed position of optimazed Trajectory 2. The black arrow shows direction of the movement.

Figures 5.11 and 5.12 show the velocity and acceleration of the first joint. The evaluation of velocity and acceleration of original trajectory has the same character as the velocity and acceleration of previous original trajectory.

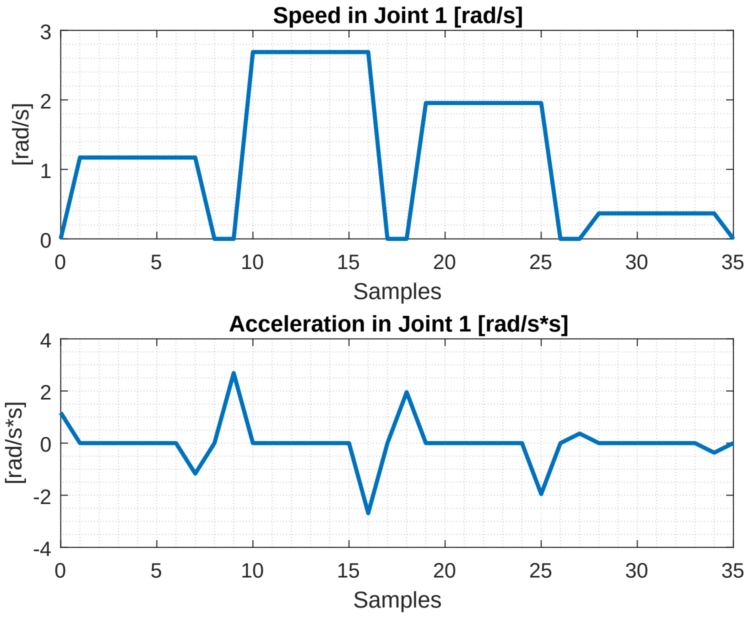

Figure 5.11: Close look up at speed and acceleration of first joint - original trajectory

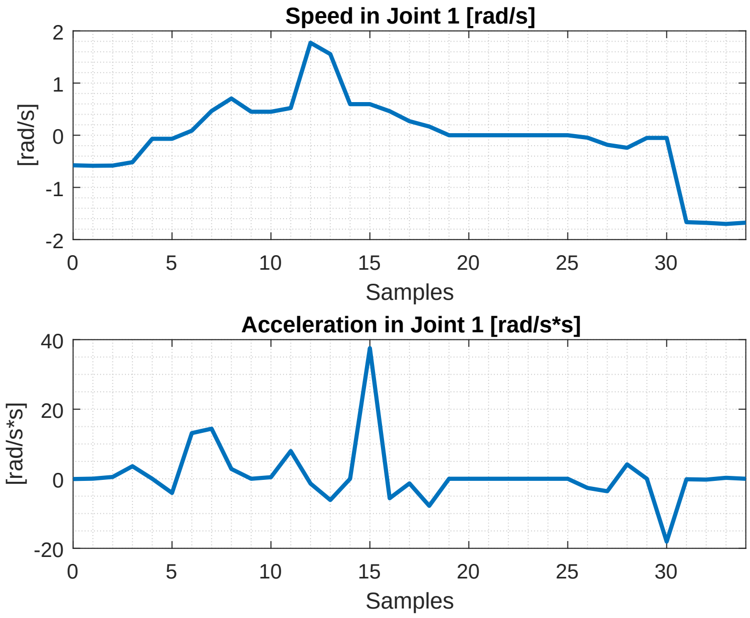

Figure 5.12: Close look up at speed and acceleration of first joint - Optimal one

Figures 5.13 and 5.14 show the power consumption of manipulator before and after optimization. Also, like in the previous section, the evolution of power consumption dependents on the acceleration.

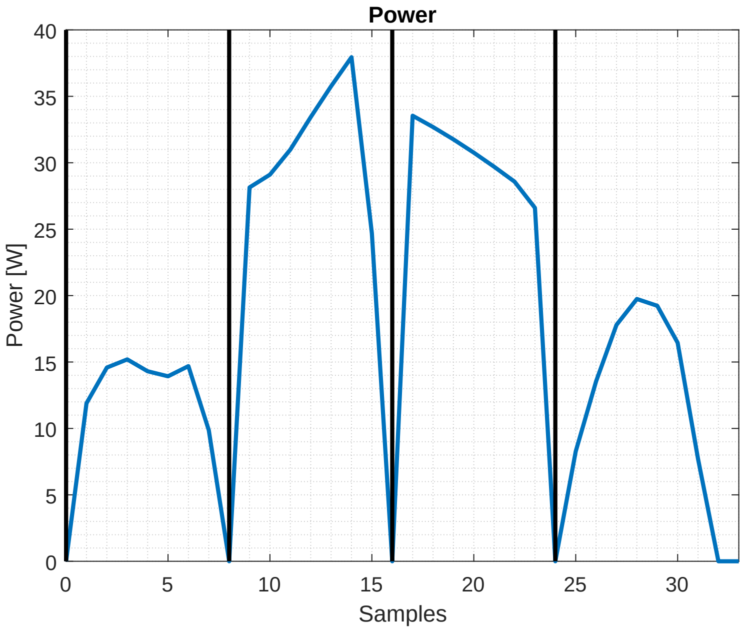

Figure 5.13: Computed power in each discretization step - original trajectory 2

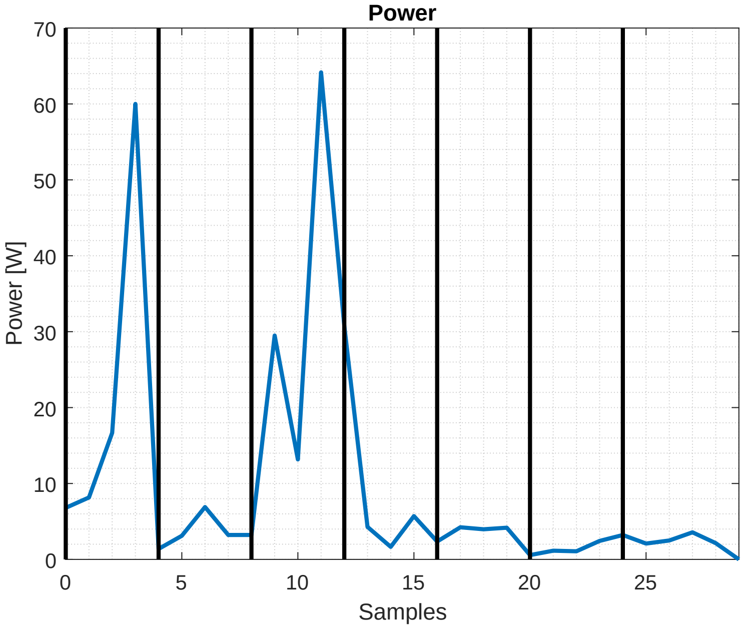

Figure 5.14: Computed power in each discretization step - optimal trajectory 2

5.2 Experiment with real manipulator

As it was mentioned in the introduction of chapter 5 the code in KRL is the output of the application. The KRL is a abbreviation for Kuka Robotic Language. Every noncooperative robot from KUKA is programmed in this language. Every program has two files: *.src* and *.dat*. The *.src* file contains the instruction for the robot. It does not contain the information about the coordinates of points. This information is held in file *.dat*. The KRL uses many types of motion command such as PTP (point-to-point), LIN (linear motion type), etc (for more information see KUKA Roboter Group GmbH [9]).

The motion is then performed by command PTP completed by the name of the point. For example, the command PTP P1 means: go to the position P1 and use point to point motion type. It is used because this motion type enables to set upper boundaries for joint speed and acceleration.

The position could be written in two ways. First one uses Cartesian notation. The point then has the coordinates X, Y and Z plus addition Euler's angles A, B and C. These angles were described in section 3.3.2. The second way uses the joint space coordinates. The points have six variables - one for each joint of the manipulator in degrees. So the positions of calculated points in radians are converted into the degrees.

Unfortunately, in KRL the exact setting velocity and acceleration for each joint isn't available. Only upper bounds for velocity and acceleration can be set in KRL. Therefore the user isn't sure whether the values computed by the application are correctly interpreted in the robot. The user is sure that the boundaries are not exceeded.

Another problem is that the velocity in PTP is calculated by the controller in percentage. The velocity could be set from 0 to 100 percent, and 100 percent means the maximal velocity of the slowest joint, which movement takes the longest duration. The calculation is the same as equations (5.1) and (5.2). Controller finds the slowest joint and adjusts the velocities for other joints. When equations(5.1) and (5.2) were used for computed values by application, every joint has the same time duration of the movement. It is caused by variable s. As it was said in chapter 4 variable s is uniquely located in time. Therefore the computed values are set as speed boundaries in KRL code.

It also has to be noted that the end-effector has to wait for 0.2 seconds in each via-points of both the original trajectories. The results presented in following sections have to be taken as the potential of energy consumption reduction.

5.2.1 Trajectory 1

The first trajectory presented in previous section 5.1 was tested on the real robotic manipulator. The power consumption was measured on WAGO-I/O-SYSTEM 750 card. This card records and processes all relevant measurement quantities in a three-phase power supply. The measured quantity was the consumed power on each phase of the power supply. Powers of all three phase were then transformed into overall power consumption. After that, the measured data were analyzed by the program (*Prumĕrná Spotřeba script*) written by Martin Ron, the supervisor of the thesis. In this program, the trajectory pattern of data is chosen. This pattern is then found with specified accuracy in the measured data, and then it was processed. Originally the pattern was identified using the peaks in the data as features.

In this thesis, the analytical program (*Prumĕrná Spotřeba script*) was improved by interpolating the data and taking all the interpolated sample values as features. With the interpolation of pattern, the start and the end times of each cycle are found in data from the measurement. Corresponding powers are then found in measurement data of power consumption Ron [12]. This approach brings more accuracy into the pattern finding and calculation of energy consumption. To have the more precise result, the same movement was performed 15 times. After that measured data was analyzed and the results are in Table 5.5.

Radius $[m]$	Final Time $[s]$	Final Energy $[Wh]$	Final Energy $[J]$	Energy Save [%]	Time Save [%]
fine	1.4	0.005628	20.2608	0	0
0.05	1.6	0.005589	20.1204	0.69	-14.3
0.1	1.4	0.004993	17.9748	11.28	0
0.15	1.4	0.005132	18.4752	8.81	0
0.2	1.4	0.004958	17.8488	11.90	0
0.25	1.6	0.005272	18.9792	6.32	-14.3
0.3	1.6	0.004813	17.3268	14.48	-14.3
0.35	1.6	0.005133	18.4788	8.79	-14.3

Table 5.5: Values from experiments with real manipulator KUKA KR5 - Trajectory 1

As it could be seen the energy consumption was reduced in the interval of 0.69 to 14.48 percent of original one. From above-mentioned reasons, mainly due to the impossibility of exact setting of required velocity and acceleration the duration of optimal movements in some cases prolonged.

Following Figures shows the measured power consumption. The Figures present measured data from original trajectory and optimal trajectory of the zone with radius 0.2 meters.

Figures 5.15 and 5.16 show the power consumption of the trajectories, which were performed 15 times. In Figure 5.15 the highest peaks are caused by returning of the robot to the start position. The sample period is 0.04 seconds.

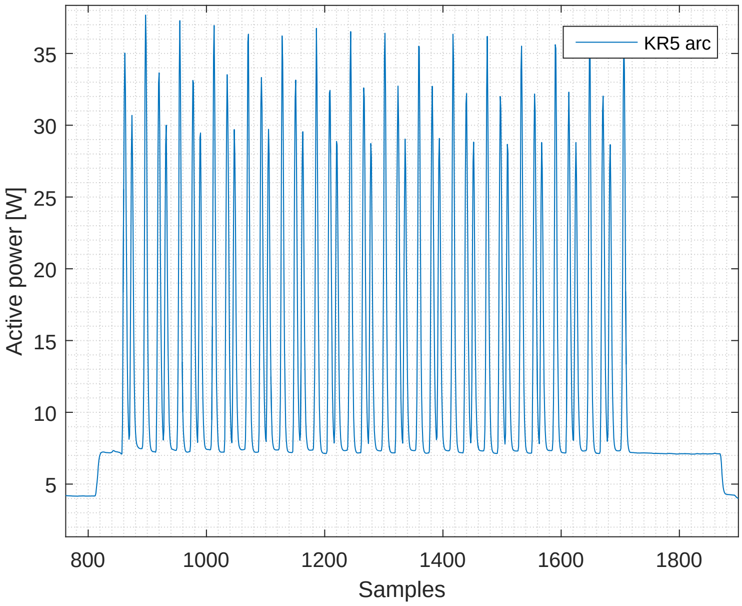

Figure 5.15: Measured data of trajectory 1 - original one

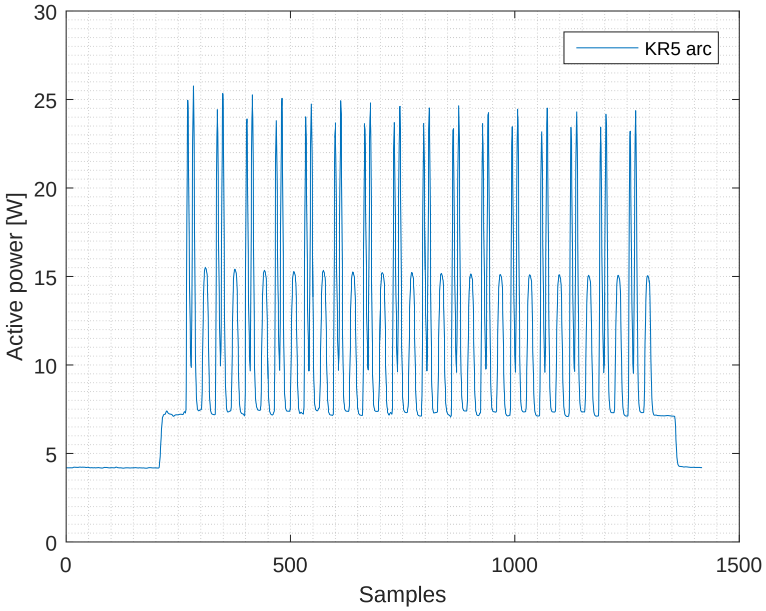

Figure 5.16: Measured data of trajectory 1 - optimal one

Figure 5.17 shows the one cycle of trajectory from the measurement. It contains the original movement and its interpolation was used for analysis in the analytical program (*Pruměrná Spotřeba script*).

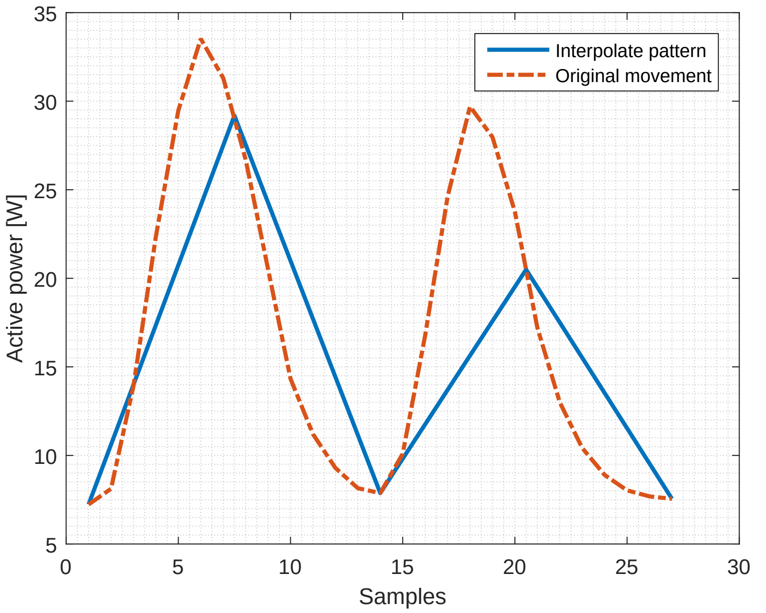

Figure 5.17: Power consumption measured on real robot and its interpolation - original trajectory 1

Figure 5.18 shows the one cycle of optimal trajectory from the measurement.

It contains the original movement and its interpolation used for analysis in the analytical program (*Pruměrná Spotřeba script*).

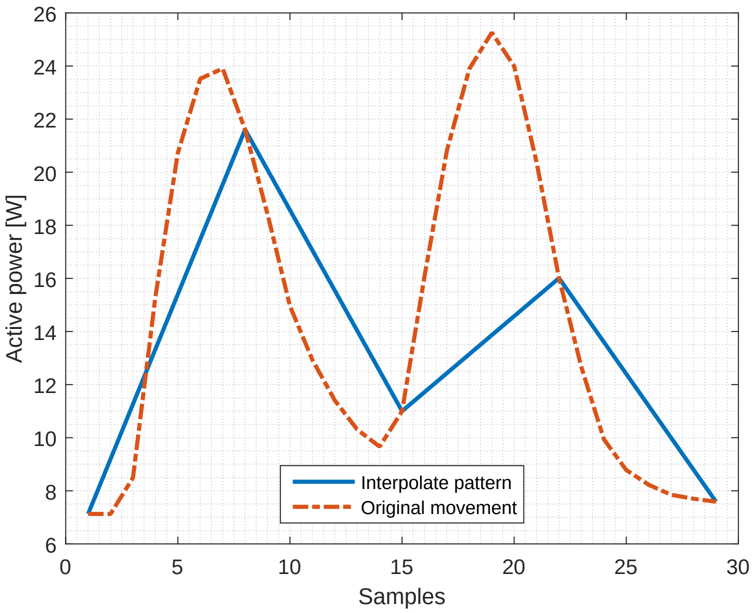

Figure 5.18: Power consumption measured on real robot and its interpolation - optial trajectory 1

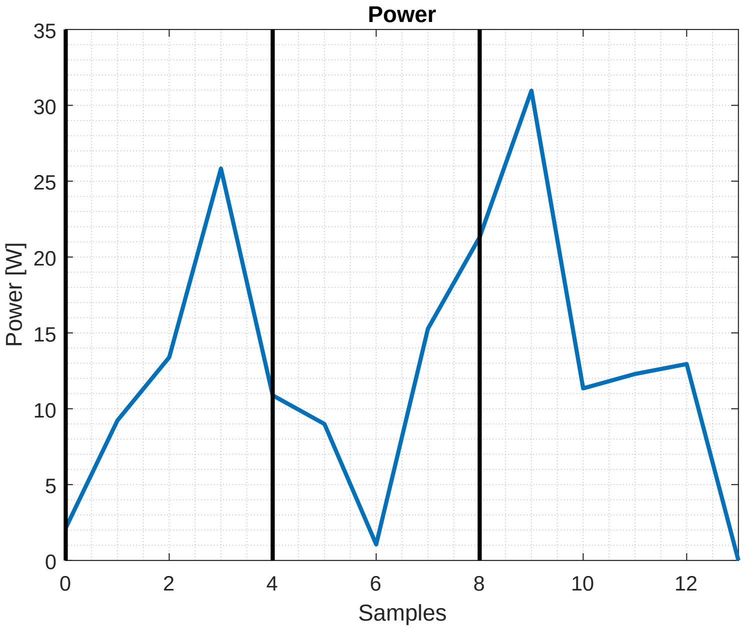

Figure 5.19: Computed power in each discretization step - optimal trajectory from simulation

As it could be seen in Figures 5.18 and 5.19, the simulated power consumption has similar behaviour as the real one.

5.2.2 Trajectory 2

The same approach as in the previous section was applied on the trajectory two. The results are:

Radius $[m]$	Final Time $[s]$	Final Energy $[Wh]$	Final Energy $[J]$	Energy Save $[\%]$	Time Save $[\%]$
fine	2.6	0.013799	49.676	0	0
0.05	2.4	0.013265	47.754	3.86	7.69
0.1	2.8	0.010966	39.4776	20.53	-16.67

Table 5.6: Values from experiments with real manipulator KUKA KR5 - Trajectory 2

Figures 5.20 and 5.21 show the measured data. Figures 5.22 and 5.23 show closer look at one cycle.

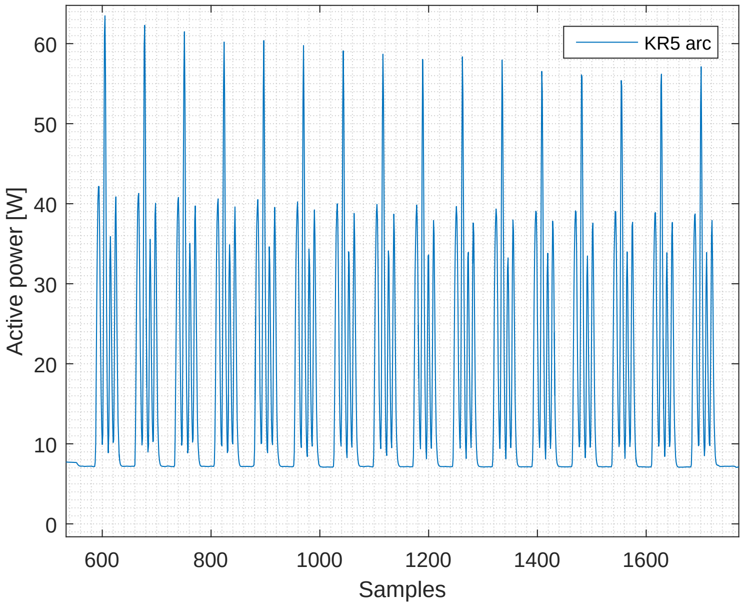

Figure 5.20: Measured power consumption of original second trajectory

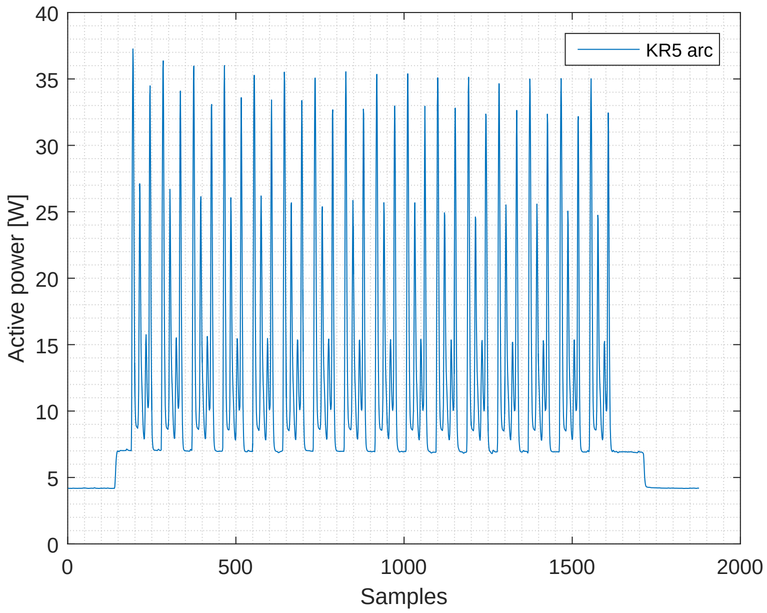

Figure 5.21: Measured power consumption of optimal second trajectory

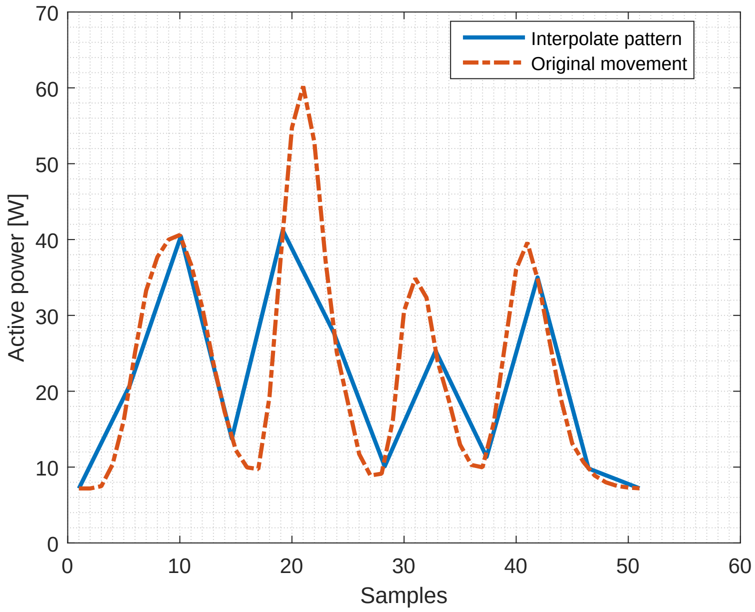

Figure 5.22: Closer look at power consumption one cycle of second original trajectory

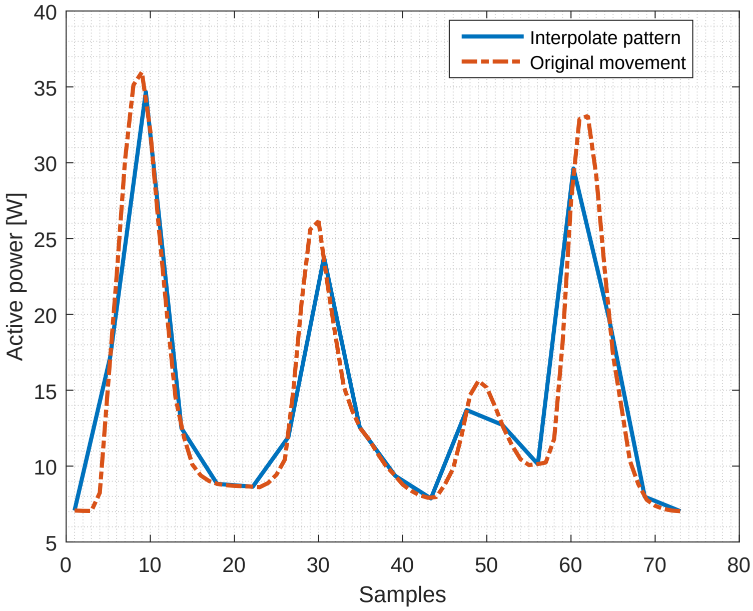

Figure 5.23: Closer look at power consumption one cycle of second optimal trajectory

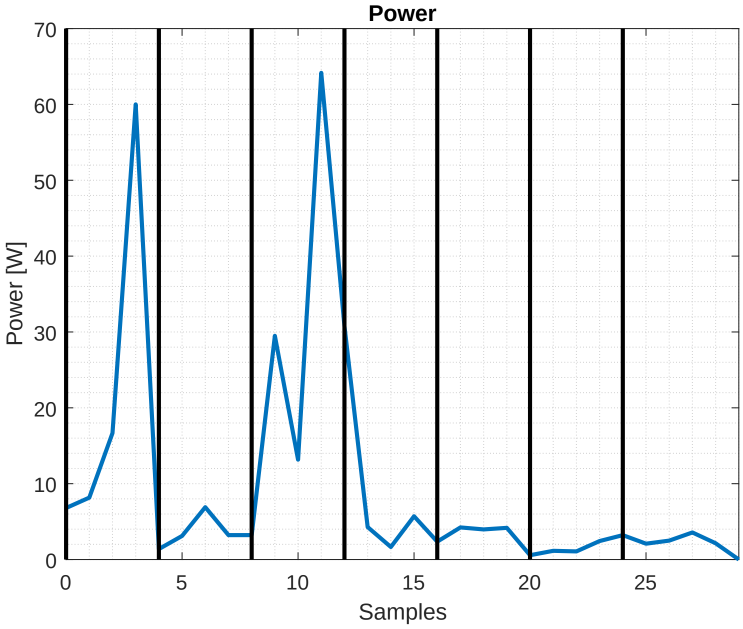

Figure 5.24: Computed power from application - Trajectory 2

In Figures 5.23 and 5.24 are differences in second part of the movement. In the simulation, power consumption is computed by the torque and if one or more joints aren't moving the torque on that joints is minimal. However, on the real manipulator the power consumption is still going on. Nevertheless, the model is still usable and produces applicable results.

Chapter 6

Conclusion

The objectives of the submitted thesis is the optimization of the robotic manipulator's energy consumption. The thesis comes from the state of the art of the problem. This work presents basic principles of the creation of the manipulator mathematical model. Kinematic and dynamic characteristics of the robot movement are simulated. The most suitable method was selected from recommended literature.

The mathematical model has been implemented on the six-axis manipulator (KUKA KR5 ARC), which is the frequently used type of robot. The nonlinear programming problem was set up to find out the most energy-effective trajectory. Interior Point Optimizer (IPOPT) finds the optimum by interactively solving a sequence of the optimization problems. The optimization problem is defined by the tailor-made criteria function, which is described as the difference between the energy spent on the movement and the energy produced by the movement.

The control variable of the criterial function is the internal parametrization variable s_i and its first and second time derivation, which describes all positions, velocities and accelerations of the arbitrary point of end-effector movement. The solver also has to keep the boundaries of trajectory variables, and has to fulfill the optimization constraints.

All the above mentioned conclusions were proofed by the experiments in the simulation environment and on the real robotic manipulator. A mathematical model is very sensitive to its parameters. The small change of the mass, the length of the links and the matrix of inertial tensor led to the large change in energy consumption. However, in all cases, the energy consumption was reduced.

In the simulation, the average energy saving was about 40 percent of the original trajectory criteria function result. On the real robot, there was not a possiblity of an exact adjustment of the required parameters. The optimal result was adjustable by the position parameter of end-effector only, and the speed and acceleration were adjustable only through the upper limits. For that reason, experiments are approximated to the optimal solution. Despite the complications, the average saving was about 10 percent of the original energy consumption.

The approach presented in this work shows that the control of the manipulator with the knowledge of its inner structure brings many advantages. Besides the energy consumption reduction, it extends the lifetime of the manipulators and in most cases accelerates the production cycle time.

6.1 Future improvements

This submitted application could be extended in the future. One of the most important extensions is the implementation of collision detection means. The other extension is the more accurate model of the drive unit (motor) of the manipulators. Improved application by these extensions would be implemented as the plug-in to the Process Simulate. The Process Simulate is the process software by Siemens, and it is used for simulation of the production lines. Also in the future, the application would be extended by robots made by other manufacturers such as ABB, Fanuc, etc.

Appendix A

Bibliography

[1] Björkenstam, S., Gleeson, D., Bohlin, R., Carlson, J. S., and Lennartson, B. (2013). Energy efficient and collision free motion of industrial robots using optimal control. In *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 510–515.

[2] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, Cambridge, UK ; New York, 1 edition.

[3] Bukata, L., Šůcha, P., Hanzálek, Z., and Burget, P. (2017). Energy Optimization of Robotic Cells. *IEEE Transactions on Industrial Informatics*, 13(1):92–102.

[4] Chinneck, J. W. (2015). Practical optimization: A gentle introduction. `http://www.sce.carleton.ca/faculty/chinneck/po.html`.

[5] Gleeson, D., Björkenstam, S., Bohlin, R., Carlson, J. S., and Lennartson, B. (2015). Optimizing robot trajectories for automatic robot code generation. In *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 495–500.

[6] Grepl, R. (2007). *Kinematika a dynamika mechatronických systémů*. Vysoké učení technické v Brně, Brno.

[7] Jönsson, V. and Ustyan, T. (2011). Implementation of a generic virtual robot controller – fraunhofer-chalmers centre. `http://www.fcc.chalmers.se/publication/implementation-of-a-generic-virtual-robot-controller/`.

[8] Kawajir, Y. (2014). Introduction to IPOPT: A tutorial for downloading, installing, and using IPOPT. `https://www.coin-or.org/Ipopt/documentation/`.

[9] KUKA Roboter Group GmbH (2015). *KUKA System Softeare 8.4 - Operating and Programming Instructions for System Integrators*. Augsburg, Germany, 1 edition.

[10] KUKA Roboter Group GmbH (2016). Specification KR 5. `https://www.kuka.com/-/media/kuka-downloads/imported/48ec812b1b2947898ac2598aff70abc0/spez_kr_5_arc_en.pdf`.

[11] MathWorks (2017). Anonymous Functions - MATLAB & Simulink. `https://www.mathworks.com/help/matlab/matlab_prog/anonymous-functions.html`.

[12] Ron, M. (2015). Teoretická analýza skriptu pro detekci cyklu. `https://docs.google.com/document/d/1_w0htNScSTRF-KbQwXDd7UWuylNz3NZFkyb_YGLy584/edit?usp=embed_facebook`.

[13] Shah, S. V. (2016). Recursive dynamics simulator (redysim). `http://redysim.weebly.com/`.

[14] Siciliano, B., Sciavicco, L., Villani, L., and Oriolo, G. (2010). *Robotics: Modelling, Planning and Control*. Springer Science & Business Media.

[15] Souček, P. (2002). *Vysoce dynamické pohony posuvů obráběcích strojů*. Společnost pro obráběcí stroje.

[16] Spong, M. W., Hutchinson, S., and Vidyasagar, M. (2005). *Robot Modeling and Control*. Wiley, Hoboken, NJ, 1 edition.

[17] Stienecker (2010). 10: Control of Industrial Robots. `https://drstienecker.com/tech-332/10-control-of-industrial-robots/`.

Appendix B

Figures from Experiments

B.1 Trajectory 1 - Simulation

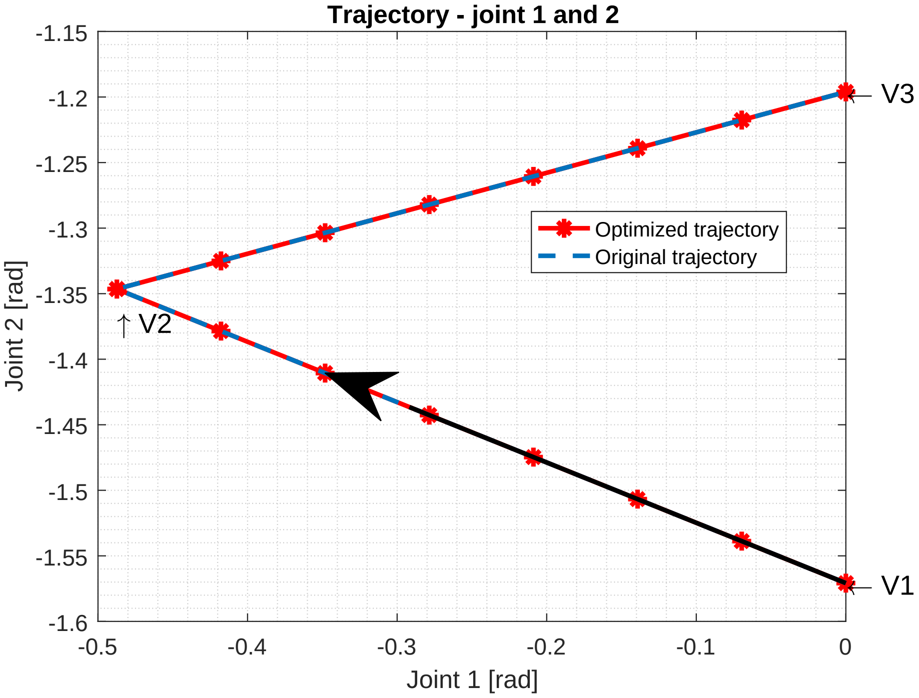

Figure B.1: Original trajectory 1 between joint 1 and 2

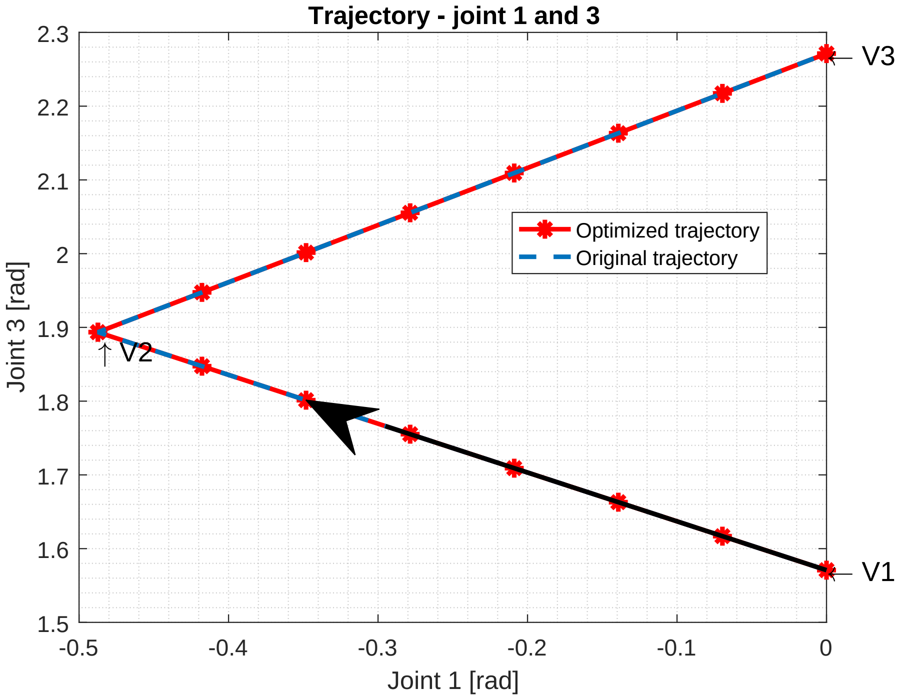

Figure B.2: Original trajectory 1 between joint 1 and 3

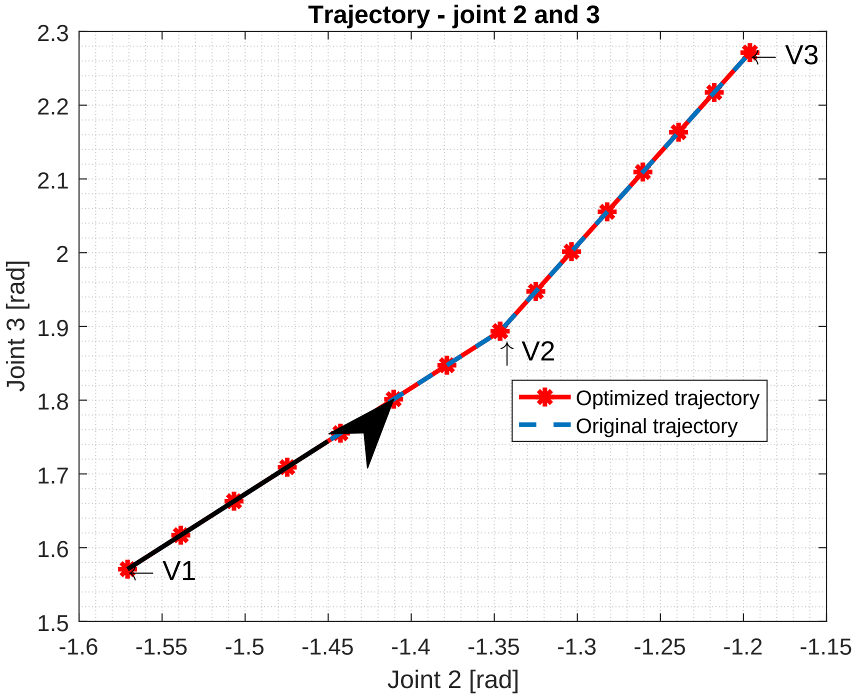

Figure B.3: Original trajectory 1 between joint 2 and 3

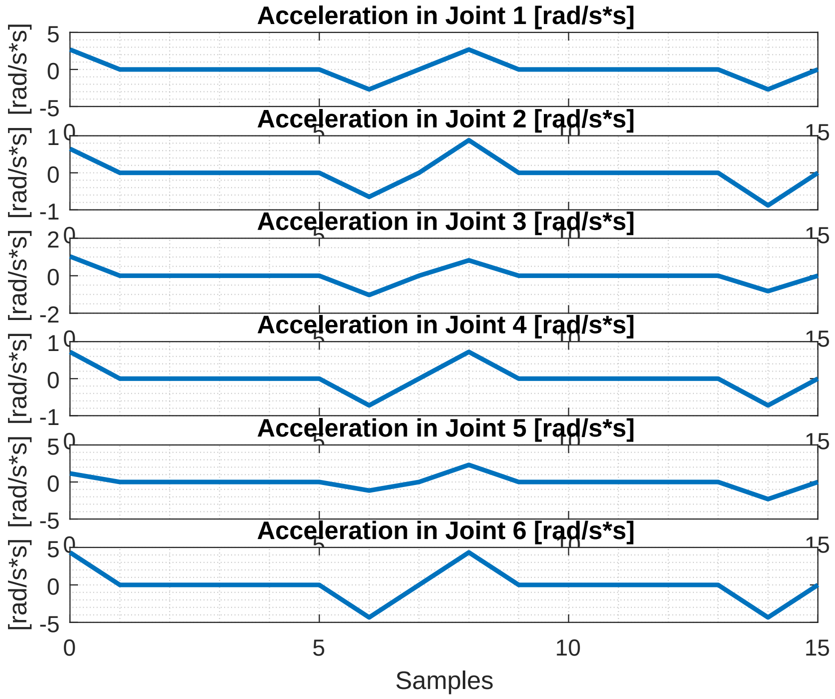

Figure B.4: Acceleration of manipulator - original trajectory 1

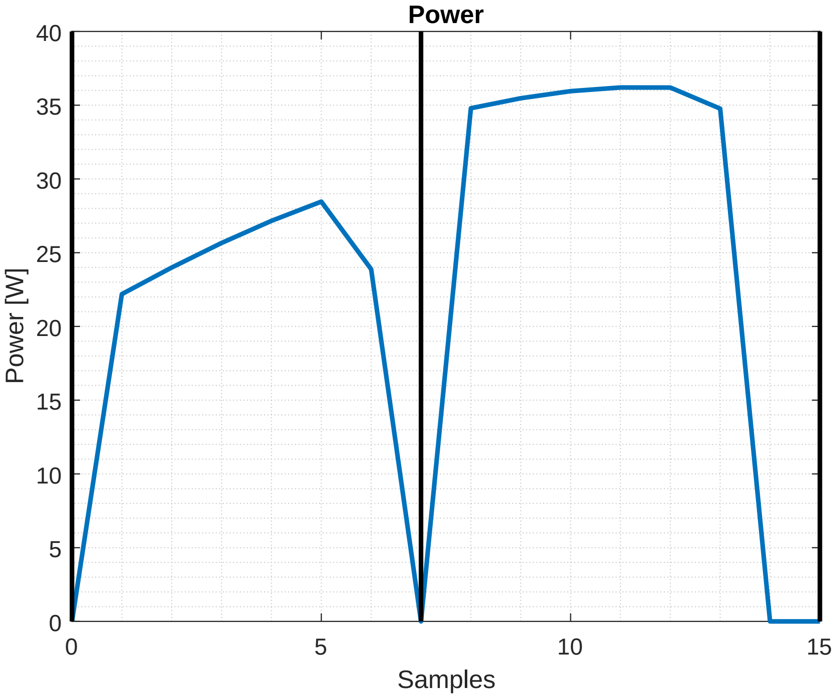

Figure B.5: Calculated power of manipulator - original trajectory 1

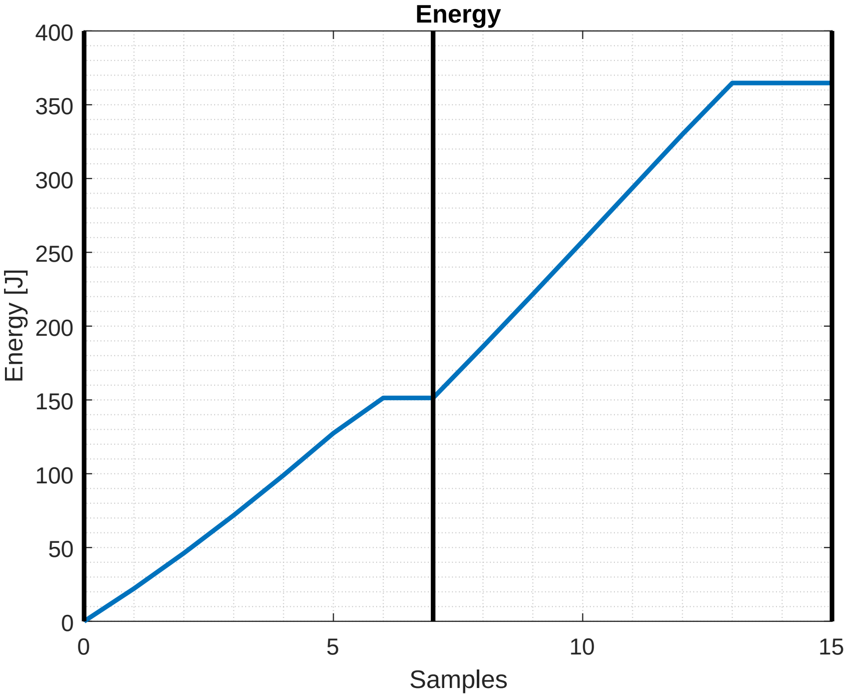

Figure B.6: Calculated energy of manipulator - original trajectory 1

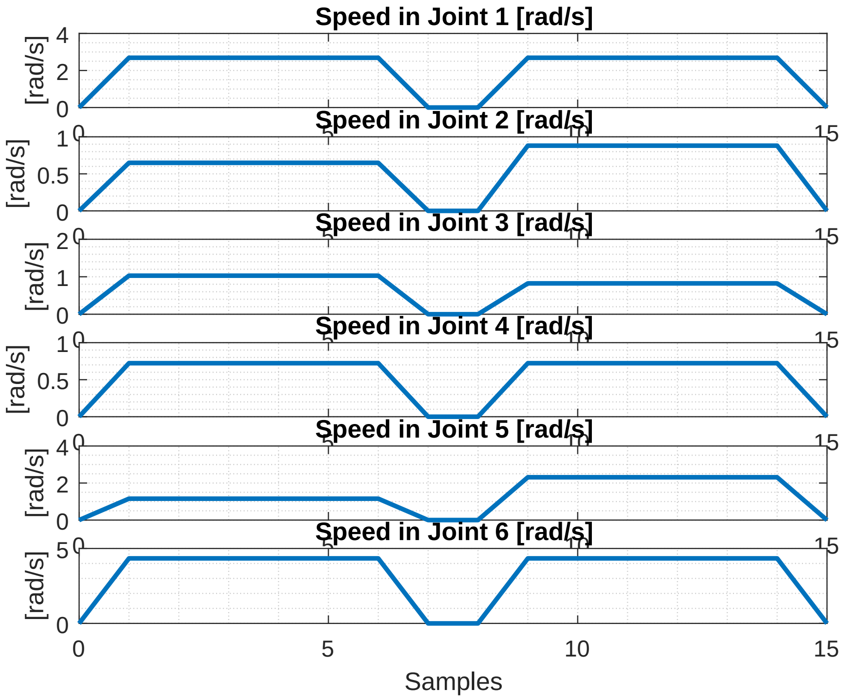

Figure B.7: Speed of manipulator - original trajectory 1

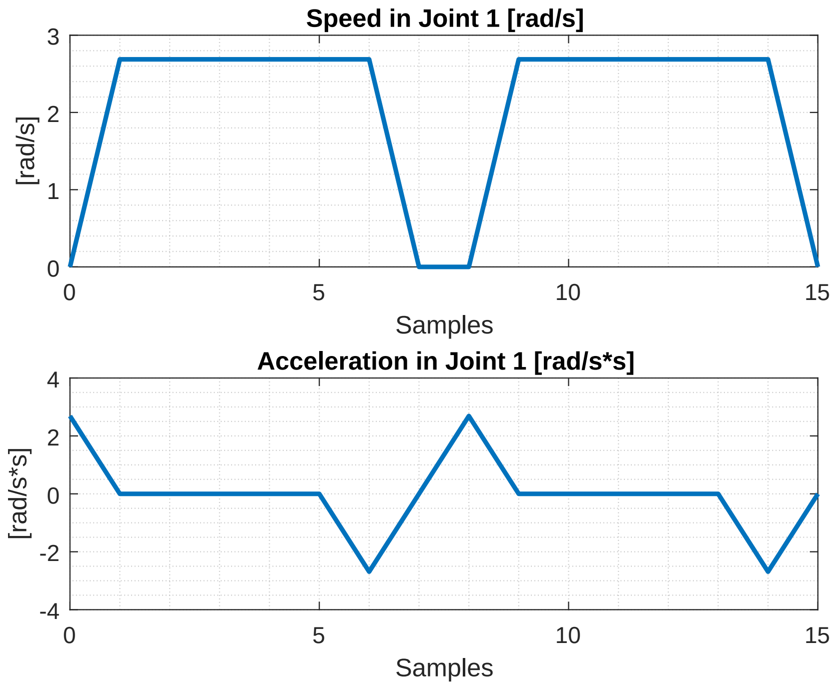

Figure B.8: Speed and acceleration of first joint - original trajectory 1

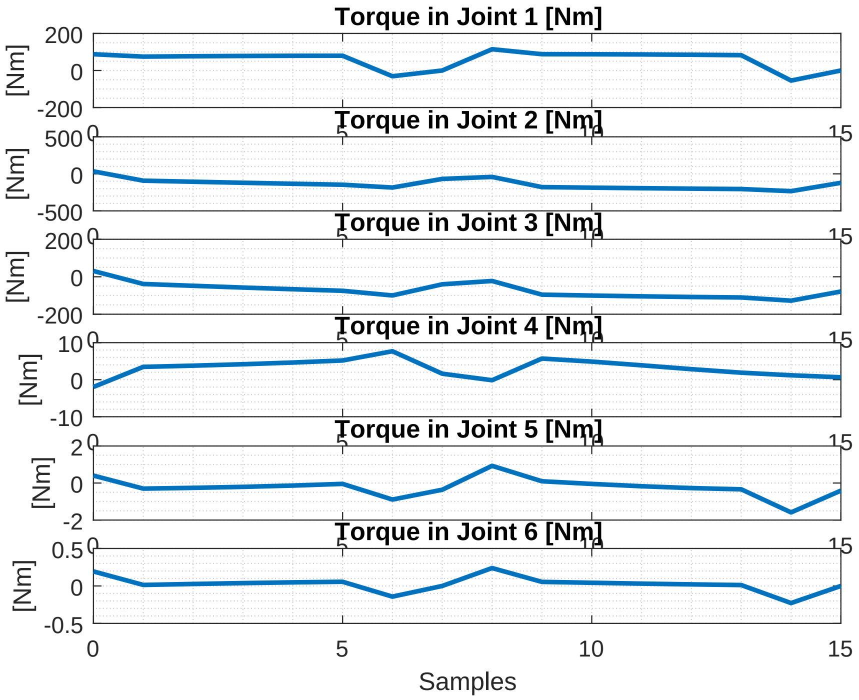

Figure B.9: Calculated torque - original trajectory 1

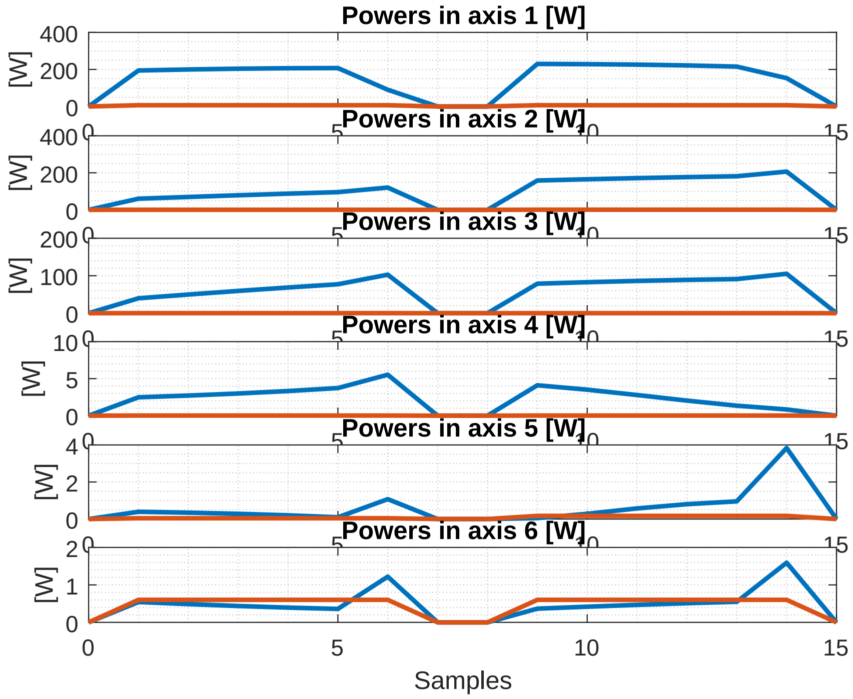

Figure B.10: Used power (Blue line) and power caused by movement (orange line) - original trajectory 1

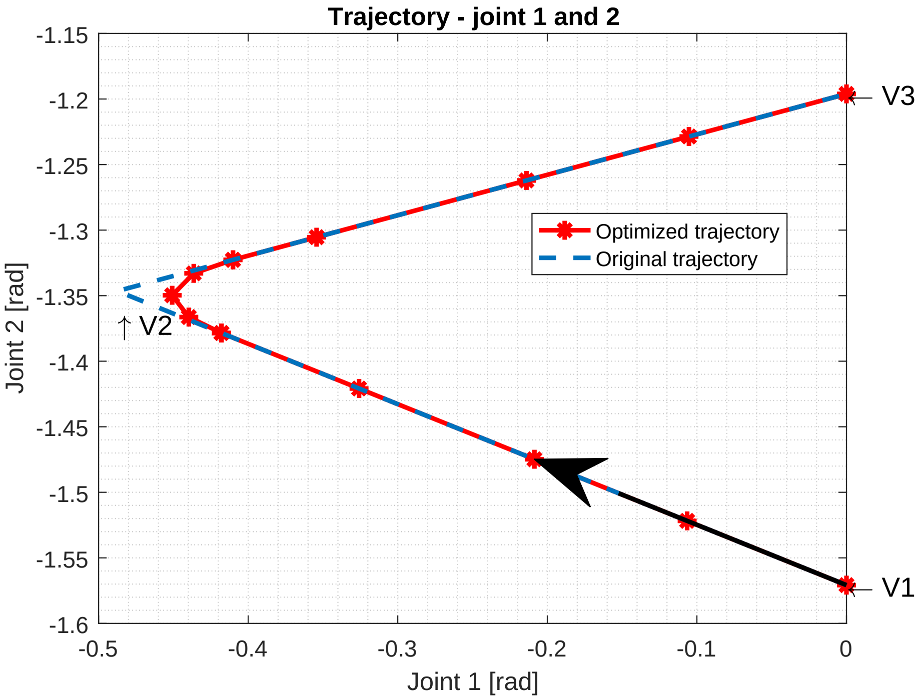

Figure B.11: Optimal trajectory between joint 1 and 2

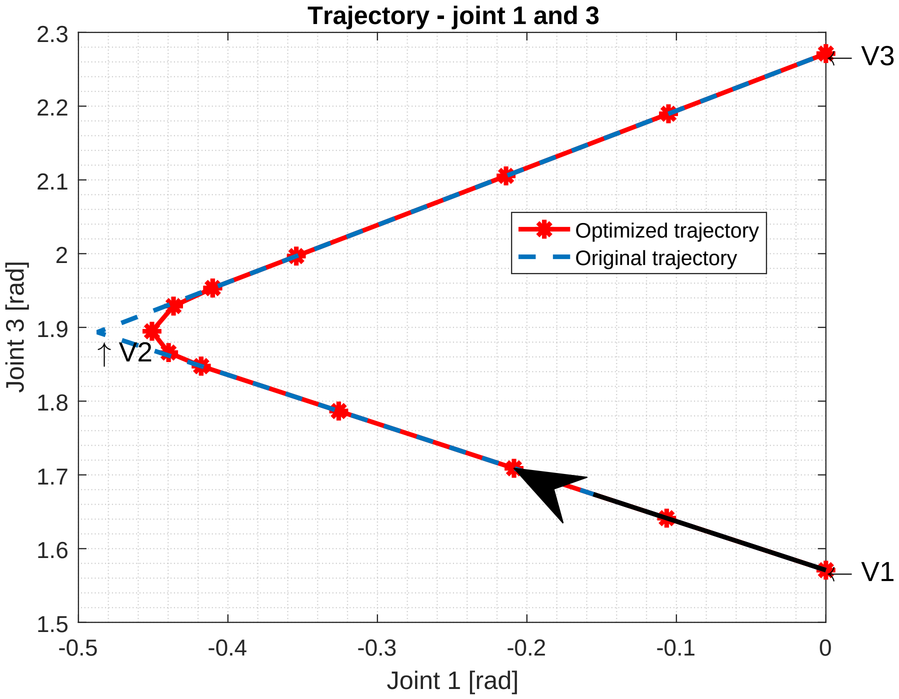

Figure B.12: Optimal trajectory between joint 1 and 3

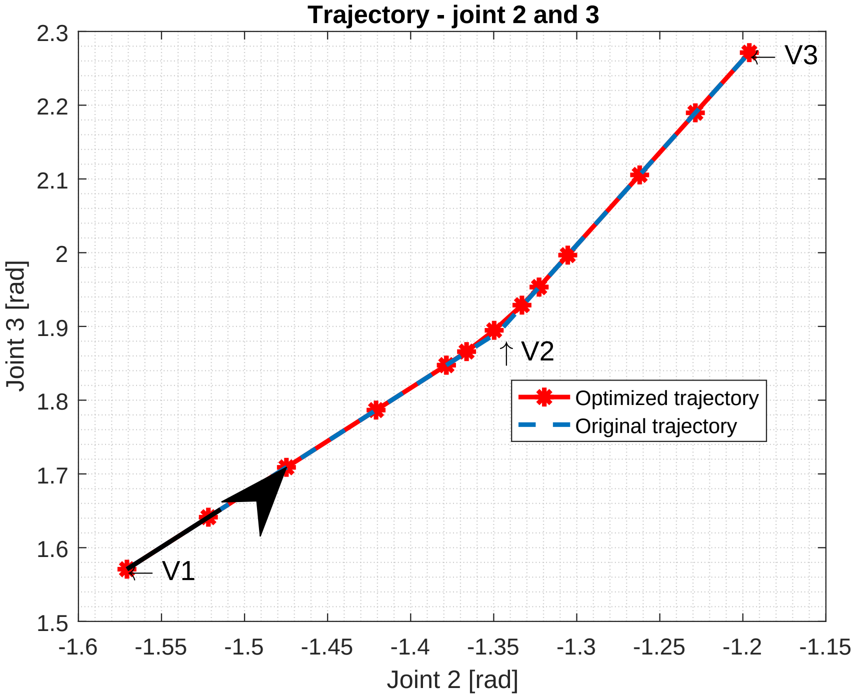

Figure B.13: Optimal trajectory between joint 2 and 3

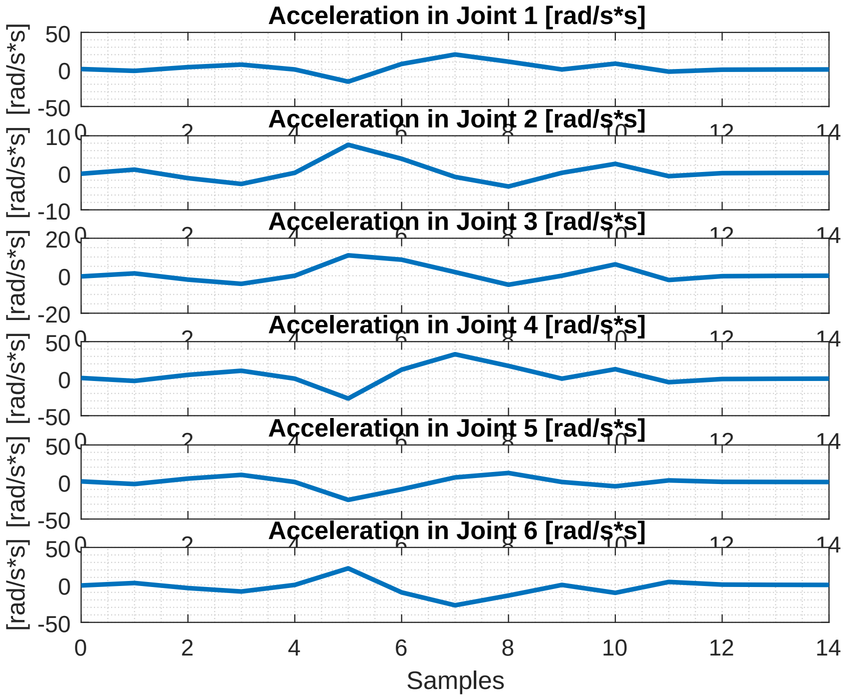

Figure B.14: Acceleration of manipulator - optimal trajectory 1

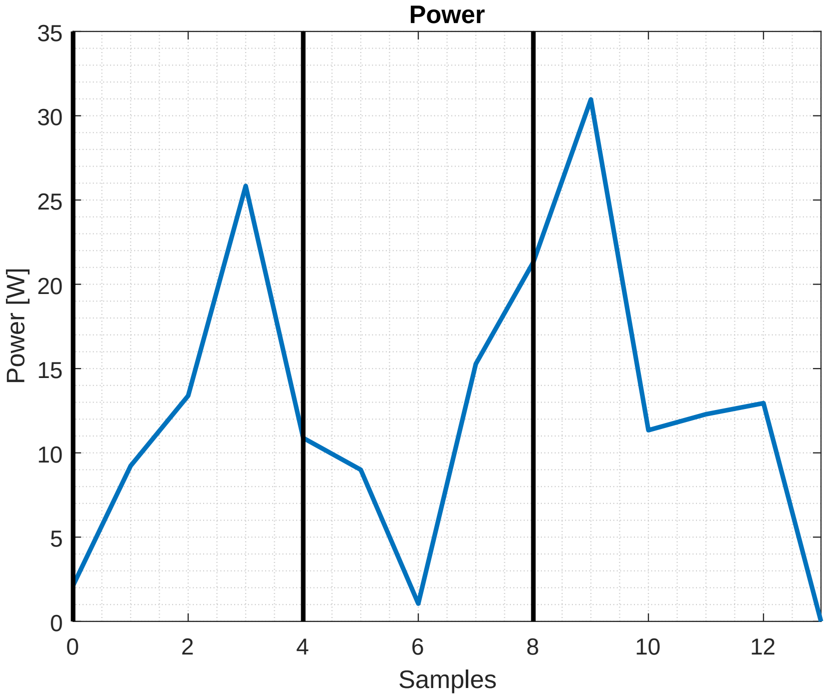

Figure B.15: Calculated power of manipulator - optimal trajectory 1

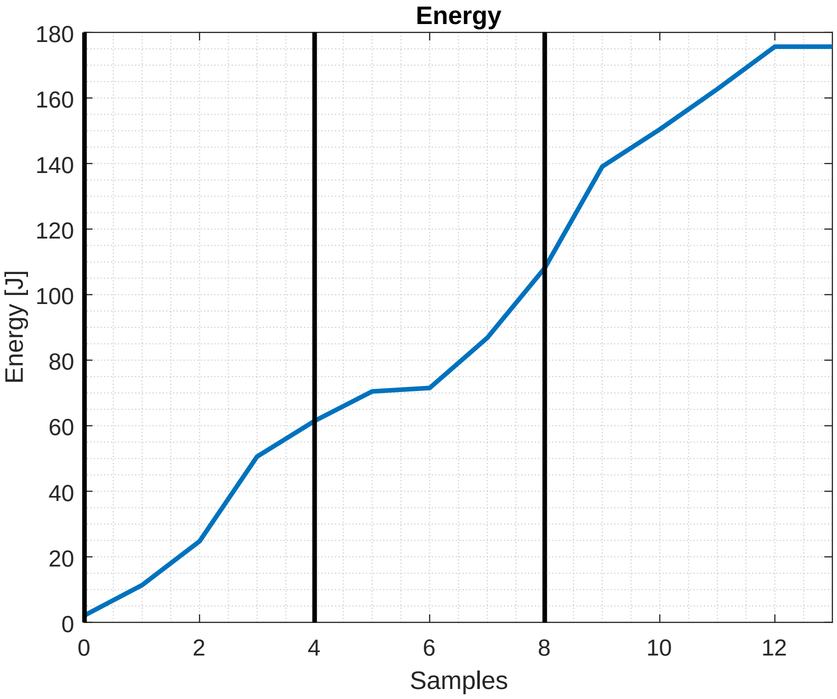

Figure B.16: Calculated energy of manipulator - optimal trajectory 1

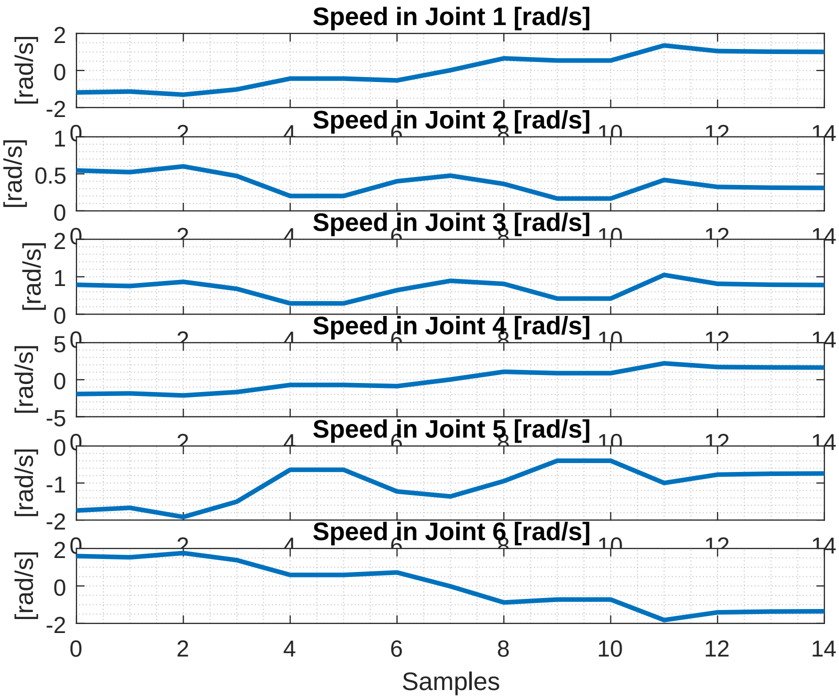

Figure B.17: Velocity of manipulator - optimal trajectory 1

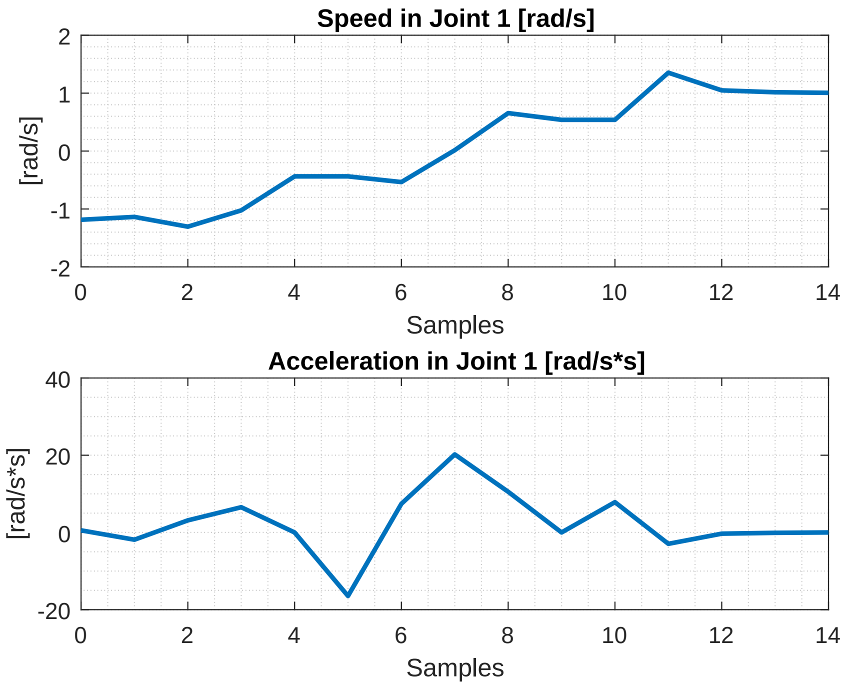

Figure B.18: Velocity and acceleration of first joint - optimal trajectory 1

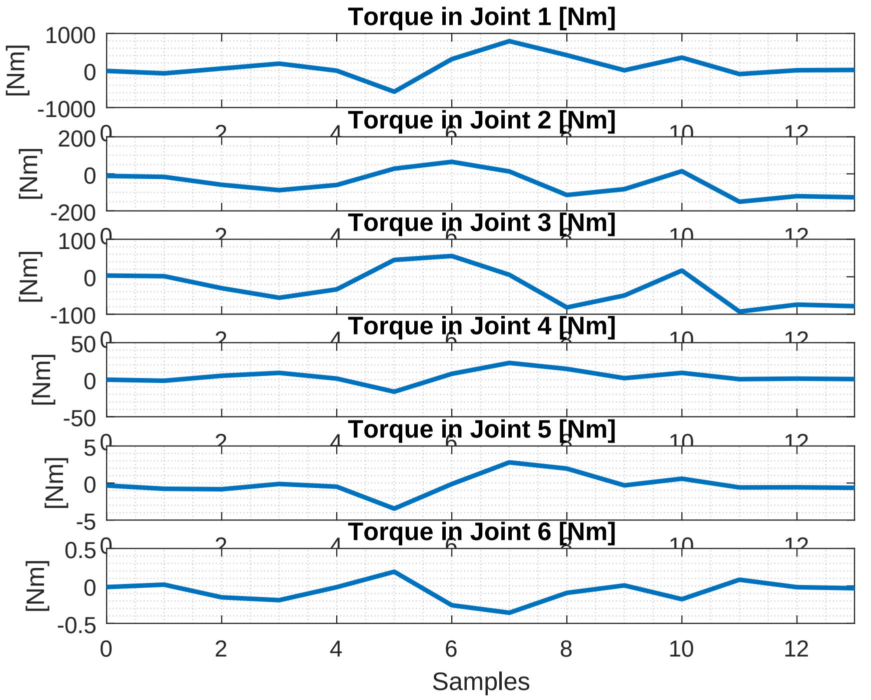

Figure B.19: Calculated torque - trajectory 1

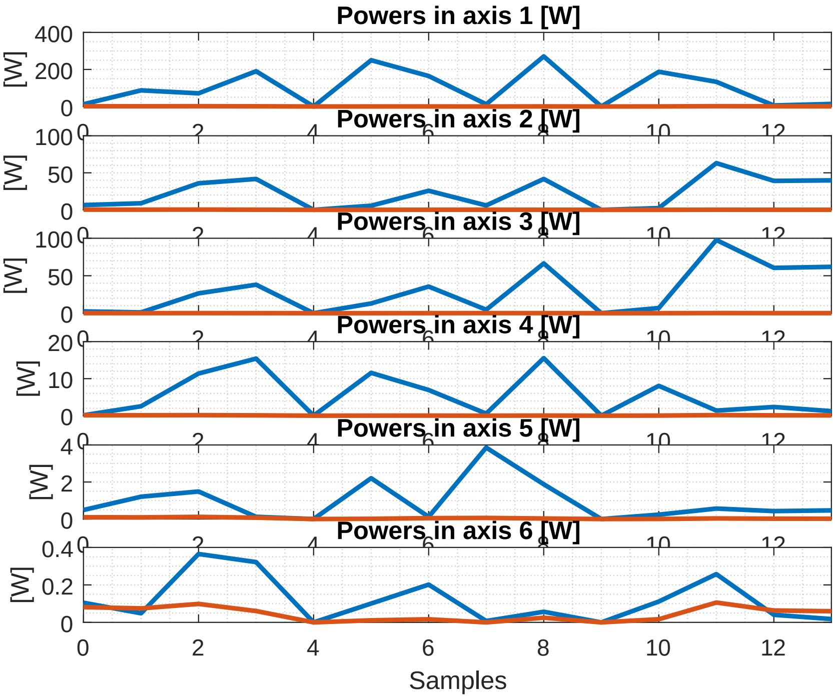

Figure B.20: Used power (Blue line) and power caused by movement (orange line) - optimal trajectory 1

Appendix C

Content of the CD

`optimization`	The folder with matlab optimization tool.
`ReDySim_Symbolic`	ReDySim algorithm scripts for generating of the dynamic equations.
`Scripts`	Matlab scripts for analysis of measured data.
`_petr_cezner_bw.pdf`	The electronic version of bachelor work.

Appendix D

Project Specification - English Version

1. Study the general principles of modeling of the dynamic movement of robotic manipulators for the purpose of its energy consumption optimization.
2. Focus on 6 axis robotic manipulators.
3. Study the recommended literature about algorithms for robotic trajectory optimization.
4. Implement studied optimization algorithms.
5. Verify the implemented algorithms using simulation.

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Cezner Petr**

Studijní program: Kybernetika a robotika
Obor: Systémy a řízení

Název tématu: **Optimalizace robotických trajektorií**

Pokyny pro vypracování:

1. Nastudujte principy modelování dynamiky pohybů robota s cílem minimalizovat jeho spotřebu.
2. V rámci práce se zaměřte na 6-osého robota.
3. Seznamte se v doporučené literatuře s algoritmy pro optimalizaci trajektorie robota.
4. Implementujte prozkoumané algoritmy optimalizace.
5. Ověřte implementované algoritmy za použití simulace.

Seznam odborné literatury:

[1] D. Gleeson, S. Björkenstam, R. Bohlin, J. S. Carlson and B. Lennartson, 'Optimizing robot trajectories for automatic robot code generation,' 2015 IEEE International Conference on Automation Science and Engineering (CASE), Gothenburg, 2015, pp. 495-500. doi: 10.1109/CoASE.2015.7294128.
[2] S. Björkenstam, D. Gleeson, R. Bohlin, J. S. Carlson and B. Lennartson, 'Energy efficient and collision free motion of industrial robots using optimal control,' 2013 IEEE International Conference on Automation Science and Engineering (CASE), Madison, WI, 2013, pp. 510-515. doi: 10.1109/CoASE.2013.6654025.
[3] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, 'Robotics: Modelling, Planning and Control,' London: Springer, 2009. ISBN: 1846286425.

Vedoucí: Ing. Martin Ron

Platnost zadání: do konce letního semestru 2017/2018

L.S.

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 9. 2. 2017